# Risk-Aware Optimization of Distribution-Based Resilient Task Assignment in Edge Computing

Hang Jin[1,2], Jiayue Liang[2], and Fang Liu[1](✉)

[1] Hunan University, Changsha, China
fangl@hnu.edu.cn
[2] Sun Yat-sen University, Guangzhou, China
{jinh26,liangjy77}@mail2.sysu.edu.cn

**Abstract.** Computing resources of mobile devices are growing, and unoccupied resources can be shared to provide support for edge computing services in edge clouds. Unlike stable servers, a significant challenge is that mobile devices may exit or join an edge cloud at any time due to change of position. This dynamic nature of mobile devices may result in abortions of task execution. In this paper, a risk-aware task assignment scheme called RATA is proposed. RATA minimizes the overhead caused by potential abortions of task execution by prioritizing tasks to the edge nodes which are unlikely to exit during task execution. We first quantify the abortion risk of each task-node pair to an expected extra overhead time, and formulate a risk-aware task assignment problem that strives to minimize the average completion time of all tasks, as well as the expected extra overhead time of each task. We then design a novel task assignment scheme to solve this problem with genetic algorithm. Finally, we implement a prototype system to evaluate the performance. The experimental results show that our scheme outperforms the state-of-art task assignment schemes in terms of average completion time and deadline miss rate in most cases.

**Keywords:** Task assignment · Task scheduling · Edge computing · Risk-aware · Mobile device

## 1 Introduction

Edge computing aims to perform computation tasks by making full use of resources at the edge of the network [1,11,16]. Similar with BOINC [2], under-utilized resources of mobile devices in crowded places such as business centers, can also be fully utilized by user devices with limited resources to speed up task execution. Assigning tasks, especially latency-sensitive ones, appropriately and efficiently in edge clouds is of great importance. However, the dynamic nature of mobile devices is a great challenge, as they can dynamically exit the edge cloud, resulting in abortions of tasks assigned to these devices. Task completion can be ensured through re-assignment and re-execution, but this brings significant

Y. Lai et al. (Eds.): ICA3PP 2021, LNCS 13157, pp. 261–277, 2022.
https://doi.org/10.1007/978-3-030-95391-1_17

overhead, resulting in missing the deadline, which is a potential time limit for a device to complete an assigned task.

Most previous works on the task assignment considered bandwidth constraints and task constraints [12,22]. However, these work did not take into account the instability of edge nodes in edge clouds as well as the unstable connectivity of mobile networks, assuming that the execution process is fault-free, which is unrealistic. Femtocloud* [10] proposes a risk-controlled task allocation mechanism, which aims to minimize the risk of edge nodes leaving before tasks are completed. However, it performs worse in terms of average completion time especially when the workload is light, compared with other approaches to minimize the average completion time.

We focus on the abortion of task execution caused by unstable edge nodes, or abruptly interrupted wireless network in edge environment. The key to solve this problem is minimizing the potential overhead time caused by abortions. Main contributions of this work can be summarized as follows.

– We study the task assignment in edge cloud consisting of dynamic mobile devices. We explicitly consider the impact of abortions of task execution on task completion time. We analyze and quantify the expected extra overhead time introduced by abortions based on the probability distribution of task completion time and the remaining presence time of edge nodes.
– We propose RATA, a risk-aware task assignment scheme. With extra overhead time to be taken into account, RATA aims to minimize the average completion time as well as the extra overhead time of each task, by selecting more reliable and powerful edge nodes for each task.
– We implement a prototype system. The experiment results show that RATA performs better in terms of average completion time and deadline miss rate, compared with the state-of-the-art works.

## 2 Related Work

Most existing works [3,12,19] focus on task assignment problem with edge servers in edge clouds. Even though they take various factors (e.g., users' location, network capabilities, order between assigned tasks) into account, execution environments of these works are failure-free. BGTA, CoGTA and TDBU [21–23] are proposed to assign social sensing tasks to mobile edge devices. These frameworks allow edge devices to choose task with preferences by game theory so as to optimize both their payoffs and the deadline miss rate. However, these works don't consider the dynamic churn rate of mobile devices.

Deng et al. [7] present a novel offloading system to make robust offloading decisions with a trade-off fault-tolerance mechanism, which can pick a better choice between waiting for reconnection and directly restarting the service when a fault happens. Although this work considers the unstable connectivity of mobile networks, it only considers reducing the overhead brought by happened abortion failures rather than reducing the occurrence of abortion failures. Habak et al. [9] present Femtocloud, an edge system to leverage mobile devices to provide edge service with a task assignment approach which aims to maximize cluster's overall useful computations. They present an improvement on the system architecture and workload management in Femtocloud* [10]. This work presents a risk-controlled task assignment approach, which is the first to focus on reducing potential abortions. It applies a point estimate (e.g., mean of subset of historical running times) to predict task completion time. Besides, Femtocloud calculates the abortion probability to represent the abortion risk. However, the abortion probability doesn't accurately reflect the extra time consumed by abortions since duration of task execution before the abortion is different. As a result, Femtocloud actually prioritizes on reducing potential abortions more than minimizing task completion time.

Tumanov et al. [17] present a novel black-box approach called JamaisVu to predict job running time utilizing its running history. This approach is tested to perform well for predicting real-world job mixes and help to make robust scheduling decisions. Park et al. [14] conduct experiments to show that the approach derived from JamaisVu can give more accurate prediction for job running time than a point estimates. Their work develops JamaisVu and presents 3Sigma which leverages full distribution of relevant running time histories to schedule jobs by using probability density function to model the utility of jobs. With this perspective of distribution-based scheduling, we quantify the abortion risk as the expected extra overhead time based on the distributions of completion time taken by tasks and the presence time that the edge nodes exist in the edge cloud.

Checkpointing can provide distributed systems [5,6] with the ability of fault tolerance by periodically saving tasks' state for recovery. Some works use checkpointing approach to build fault-tolerance mechanisms for robust computation offloading [7,8]. Since we mainly focus on the intrinsic ability of task assignment schemes, we don't apply checkpointing into RATA and baselines for task recovery.

## 3 System Model and Problem Formulation

### 3.1 System Model

A diagram of edge cloud leveraging mobile devices is shown in Fig. 1. The model includes following components.

**Edge Cloud:** An edge cloud consists of a controller (a cloud server or edge server) and a certain number of edge nodes (servers or mobile devices), connected with each other via Access Points (APs) in a Local Area Network (LAN). Presence time is used to denote the duration between the moment a mobile device joins and the moment the mobile device exits. Edge nodes are modeled as $EN = \{en_1, en_2, ..., en_J\}$, where $J$ is the total number of edge nodes in the system. Transmission rates between the controller and edge nodes are modeled as $TX = \{tx_1, tx_2, ..., tx_J\}$. Each edge node maintains a certain number of workers to execute tasks, and a waiting queue for tasks. For edge node $j$ where $1 \leq j \leq J$, the waiting queue length of it is defined as $ql_j$, and the average execution time of tasks among workers is defined as $T_j^{Avg}$. Note that mobile devices in the figure are

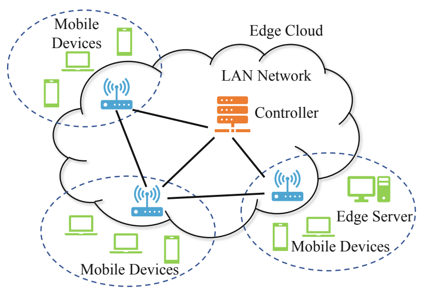


**Fig. 1.** An edge cloud leveraging mobile devices.

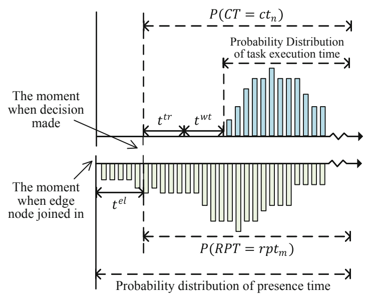

$P(CT = ct_n)$
Probability Distribution of task execution time
The moment when decision made
$t^{tr}$
$t^{wt}$
The moment when edge node joined in
$t^{el}$
$P(RPT = rpt_m)$
Probability distribution of presence time

**Fig. 2.** Relation of related probability distributions.

the ones sharing their underutilized computation resources, rather than devices submitting tasks.

**Edge Service:** An edge service is invoked by a mobile user. Each edge service accepts a task with input data, and produces a result by assigning them to edge nodes for executing. All tasks in one assignment are modeled as $TK = \{tk_1, tk_2, ..., tk_I\}$, where $I$ is the total number of tasks. For task $i$ where $1 \le i \le I$, the size of the input data is defined as $s_i$. Since edge nodes are heterogeneous in terms of hardware and software, execution time of a specified task varies from node to node. We define $t_{i,j}^{ex}$ as the execution time of task $i$ in edge node $j$ where $1 \le i \le I, 1 \le j \le J$.

**Task Assignment Strategy Profile:** A task assignment strategy profile defines each task's choice of edge node, which are modeled as variable matrix $X = \{x_{1,1}, x_{1,2}, ..., x_{I,1}, x_{I,2}, ..., x_{I,j}\}$. $x_{i,j} = 1, 1 \le i \le I, 1 \le j \le J$ implies task $i$ would be assigned to edge node $j$, while $x_{i,j} = 0$ means not.

## 3.2 Problem Formulation Without Abortion Risk

The entire process of task assignment consists of three stages. First, the task and its input data need to be transmitted from controller to the target edge node. If all the workers in the target edge node are busy, then the task will be added into the waiting queue and wait until being chosen to execute. At last, the task will be executed by one of the workers in the target edge node. Therefore, if we assume that the connection is always stable, the completion time $t_{i,j}^{cpt}$ of task $i$ in edge node $j$ can be calculated as

$$t_{i,j}^{cpt} = t_{i,j}^{tr} + t_j^{wt} + t_{i,j}^{ex} \tag{1}$$

where the transmission time $t_{i,j}^{tr}$ can be calculated by $t_{i,j}^{tr} = \dfrac{s_i}{tx_j}$. The waiting time $t_j^{wt}$ can be calculated by $t_{i,j}^{wt} = ql_j \times T_j^{Avg}$. The execution time $t_{i,j}^{ex}$ can be predicted using the distribution of task execution time history, we will describe the approach in Sect. 4. Equation (1) ignores propagation latency as well as the transmission time of the output, because they are too small to be compared with other parts.

For Eq. (1), a frequently-used task assignment policy is assigning a task to the edge node with shortest complete time greedily. However, this policy may assign multiple tasks to a particular edge node, which results in extra waiting time among these tasks. In consideration of this issue, we formulate the task assignment problem as a 0–1 integer linear program as follows, which strives to minimize the batch completion time of all the tasks, in other words, to minimize the average completion time.

$$min \sum_{i,j}(x_{i,j} \times t_{i,j}^{cpt}) + (\sum_{i} x_{i,j} - 1) \times T_j^{Avg} \tag{2}$$

$$s.t. \sum_{j} x_{i,j} = 1 \tag{3}$$

$$x_{i,j} \in \{0,1\}, 1 \leq i \leq I, 1 \leq j \leq J \tag{4}$$

We present a congestion-avoiding mechanism which adds up a compensation term to represent the extra waiting time cause by multiple tasks in the same edge node, in order to solve the aforementioned issue. Constraints (3) ensure that each task can be only assigned to one edge node.

## 3.3 Problem Formulation with Abortion Risk

To consider the dynamic nature of mobile devices and the abortion risk, we follow the perspective of distribution-based scheduling [14]. We think of the completion time of task, as well as the presence time of edge nodes using probability distribution. In fact, we focus on the remaining presence time which is equal to the difference between the presence time of an edge node and the elapsed time from the moment it joined in. When assigning a task to an edge node, we suppose the remaining presence time of the edge node as random variable $RPT$ and its law of probability distribution is

$$p_m = P(RPT = rpt_m), m = 1, 2, ..., M \tag{5}$$

where $rpt_m$ denotes the possible values of $RPT$ and $M$ denotes the total number of these values. Similarly, we suppose the completion time of the task as random variable $CT$ and its law of probability distribution is

$$q_n = P(CT = ct_n), n = 1, 2, ..., N \tag{6}$$

where $ct_n$ denotes the possible values of $CT$ and $N$ denotes the total number of these values. In fact, the two random variables are independent of each other. The relationship of related probability distribution is showed in Fig. 2, where $t^{el}$ refers to the time elapsed from the moment edge node joined in.

Then we can calculate the abortion probability $\alpha$, namely the probability that the remaining presence time is shorter than the completion time, by Eq. (7).

$$\alpha = P(RPT \leq CT) = \sum_{rpt_m \leq ct_n} (p_m \times q_n) \tag{7}$$

Further, we quantify the extra time that a potential abortion of task execution will bring up. Since the task assignment decisions at each cycle are independent with each other, in order to calculate the potential abortion overhead for one certain decision, we assume that after an abortion, the second execution of the task will be certainly completed. And we assume this task will be reassigned to an edge node which has a similar capacity with that of the origin one, so the completion time will be the same. Based on these assumptions, we suppose the total completion time under abortion risk as a random variable $TCT$ defined by Eq. (8).

$$TCT = \begin{cases} CT & , RPT > CT \\ RPT + CT & , RPT \leq CT \end{cases} \tag{8}$$

The expectation of $TCT$ can be calculated by:

$$\begin{aligned} E(TCT) &= \sum_{rpt_m > ct_n} (p_m \times q_n \times ct_n) + \sum_{rpt_m \leq ct_n} (p_m \times q_n \times (rpt_m + ct_n)) \\ &= \sum (p_m \times q_n \times ct_n) + \sum_{rpt_m \leq ct_n} (p_m \times q_n \times rpt_m) \\ &= E(CT) + \sum_{rpt_m \leq ct_n} (p_m \times q_n \times rpt_m) \end{aligned} \tag{9}$$

where $E(CT)$ is the expectation of $CT$. We define random variable $ET$ as the extra time caused by the abortion, the expectation of $ET$ will be

$$E(ET) = \sum_{rpt_m \leq ct_n} (p_m \times q_n \times rpt_m) \tag{10}$$

Particularly, $E(ET)$ will be zero if the abortion probability $\alpha$ equals to zero, when an edge node will never exit from the system or lose connection to the controller. We can treat the expected extra time as the representative of the abortion risk. The shorter the expected extra time is, the smaller the abortion risk is.

Based on the derivation above, we update the Eq. (1) to Eq. (11). We define $t^{et}_{i,j}$ as the predictions of the expected extra time $ET$. Therefore, the expected task completion time $t^{ect}_{i,j}$ under abortion risk, is the sum of task completion time and the expected extra time.

$$t^{ect}_{i,j} = t^{cpt}_{i,j} + t^{et}_{i,j} \tag{11}$$

We will describe the details to predict $t_{i,j}^{et}$ in Section 4. The risk-aware task assignment problem can be updated as follows.

$$min \sum_{i,j} (x_{i,j} \times t_{i,j}^{ect}) + (\sum_{i} x_{i,j} - 1) \times T_j^{Avg} \tag{12}$$

$$s.t. \sum_{j} x_{i,j} = 1 \tag{13}$$

$$x_{i,j} \in \{0,1\}, 1 \leq i \leq I, 1 \leq j \leq J \tag{14}$$

This updated problem aims at minimizing the sum of all the expected task completion time. In the optimization process, the solver will try to choose a stabler edge node to reduce the expected extra time for each task.

## 4 System Design of Task Assignment

### 4.1 System Architecture

The overview of system architecture is shown in Fig. 3. It consists of two main components: a controller and a set of edge nodes. The controller is responsible for receiving tasks from users, distributing tasks to suitable edge nodes and return the results. It consists of following modules.

- **Task Originator Interface:** receives service requests (tasks and their input data) from users, and returns back results of completed tasks.
- **Task Manager:** collects tasks, which are new or going to be reassigned, and submits these tasks to the Scheduler.
- **Node Manager:** maintains the connection with edge nodes and collects information of each edge node (e.g. IP address and waiting queue length).
- **Predictor:** responsible for generating time predictions as the input in equation (1) and (11). It consists of several Task Predictor Units and Node Predictor Units. Each Task Predictor Unit maintains distributions of execution time for tasks of a group of edge nodes with the same software and hardware configuration. Similarly, Each Node Predictor Unit maintains distributions of presence time for a group of edge nodes with the same network environment. The way to make predictions is discussed in the Subsect. 4.2.
- **Scheduler:** collects predictions of task-node pairs from the Predictor, and make assignment decisions at the start of each assignment cycle.
- **Task Tracker:** forwards tasks to target edge nodes, then tracks execution states, returns task results to the Task Manager and the Predictor. Specially, failed tasks will be kept in Task Manager for next assignment.

Edge nodes are responsible for executing their assigned tasks. Each of them consists of following modules.

- **Task Manager:** receives assigned tasks from the Controller, puts them into a waiting queue, and returns execution results after completed.
- **Work Thread:** when idle, execute a task in the waiting queue.
- **Heartbeat Thread:** responsible for keeping the connection with the Controller, report node status periodically.

## 4.2 Generating Predictions and Probability Distributions

We follow the black-box approach of $3\sigma$Predict [14] to generate distributions and predictions for tasks. This approach does assume that most tasks will be similar to some subsets of previous tasks, and similar tasks will have similar execution times. It does not require any task structure, or user-provided information about the execution time, but a set of features of each task. We choose three features, the user name of the task, the task name, and the logic task name (e.g. face recognition). For each feature-value pair, the approach tracks every completed tasks which have the same value of that feature and generate an empirical distribution which is stored as a histogram using a stream histogram algorithm [4] with a maximum of 40 bins. Each histogram will maintain four point estimates using four estimation techniques: average, median, moving average with decay 0.5 and average of 20 recent tasks. In addition, each histogram tracks the Normalized Mean Absolute Error (NMAE) of these four estimates. When predicting the execution time of a task with several features, this approach firstly collects all point estimates of these features. Then it chooses the point estimate with lowest NMAE as the prediction of execution time, and at last, the distribution which owns that point estimate will be chosen as the distribution of execution time for the task. Based on the approach described above, each Task Predictor Unit serves for a group of edge nodes with the same machine type. A machine type means a configuration of hardware and software. Edge nodes with the same machine type will have a similar computational capacity. What's more, we append one histogram to track all the completed tasks regardless of features in each Task Predictor Unit, in order to predict the average task execution time $t^{Avg}$.

As for the presence time of edge nodes, we use the same histogram algorithm to generate distributions in each Node Predictor Unit serving for a group of edge nodes with the same environment type. A environment type represents a concrete network environment. For example, edge nodes connecting to the same AP share the same type. We assume that each edge node with the same type will have similar characteristic of mobility.

We append two algorithms as follows to translate a histogram to a probability histogram, based on the histogram algorithms defined in [4]. A histogram is a set of B pairs (called bins) as an approximate representation of a set of real numbers, denoted by $\{(v_1, f_1), ..., (v_B, f_B)\}$. For each pair $(v_i, f_i), 1 \leq i \leq B$, $v_i$ denote the value of a number, $f_i$ denote the frequency of the number. We define a new concept called probability histogram as a variant of the histogram. A probability histogram is also a set of B pairs $\{(v_1, p_1), ..., (v_B, p_B)\}$, but for each pair $(v_i, p_i), 1 \leq i \leq B$, $p_i$ denotes the probability of the number. In other word, a probability histogram is an approximate representation of a probability distribution. These two new algorithms show the details of two particular translation from histogram to probability histogram. Algorithm 1 first performs a right shift operation on a histogram with an offset, then translates the new histogram to be a probability histogram. Algorithm 2 first filters the pairs whose value is smaller than an offset from a histogram, then translates the rest of the histogram to a probability histogram.

**Algorithm 1.** Right-Shift and Probability Translation Procedure

```
Input: A histogram $\{(v_1, f_1), ..., (v_B, f_B)\}$, a offset $\delta$
Output: A probability histogram $\{(u_1, p_1), ..., (u_B, p_B)\}$.
 1: Set $S = \sum_{i=1}^{B} f_i$
 2: for $i = 1$ to $B$ do
 3:     $u_i = v_i + \delta$
 4:     $p_i = f_i \div S$
 5: end for
```

**Algorithm 2.** Probability Translation Beginning with an Offset Procedure

```
Input: A histogram $\{(v_1, f_1), ..., (v_B, f_B)\}$, a offset $\delta$
Output: A probability histogram $\{(u_1, p_1), ..., (u_C, p_C)\}$.
 1: Find $i$ such that $p_i \leq \delta < p_{i+1}$
 2: if $p_i = \delta$ then
 3:     Set $S = \sum_{j=i}^{B} f_i$
 4: else
 5:     Set $S = \sum_{j=i+1}^{B} f_i, i = i + 1$
 6: end if
 7: for $j = i$ to $B$, $k = 1$ to $B - i + 1$ do
 8:     $u_k = v_j$
 9:     $p_k = f_k \div S$
10: end for
```

The relation between probability distribution of task completion time and that of task execution time is showed in Fig. 2. To get the probability histogram of task completion time, we first calculate the sum of transmission time and waiting time as the offset, then we call Algorithm 1 with histogram of task execution time as the input. Similarly, the relation between probability distribution of remaining presence time and of presence time are also depicted in Fig. 2. To get the probability histogram of remaining presence time, we first calculate the elapsed time from the moment when the edge node joined as offset, then we call Algorithm 2 with histogram of presence time as the input.

At last, once we have generated the probability histogram of task completion time and that of the remaining time of edge node, we can calculate the expected extra time by Eq. (10), and the expected task completion time by Eq. (11).

In consideration of scalability, we employ three techniques to reduce the memory footprint. Firstly, the original design of $3\sigma$Predict can maintain a histogram using constant memory regardless of the number of points. Secondly, in spite of heterogeneity, we use one Task Predictor Unit to serve for a groups of edge nodes with the same machine type rather than just one edge node. Finally, we use one Node Predictor Unit to serve for a groups of edge nodes with the same environment type rather than just one edge node.

### 4.3 Task Assignment Algorithm

We choose the genetic algorithm to solve the problem, which is a heuristic method to search approximate solutions for optimization problems with evolutionary theory, and is often used to solve task assignment problems [7,13,20].

For a given task set $TK = \{tk_1, tk_2, ..., tk_I\}$ and edge node set $EN = \{en_1, en_2, ..., en_J\}$, we encode the task assignment strategy $X = \{x_{i,j}, 1 \leq i \leq I, 1 \leq j \leq J\}$ to a vector $S = \{s_i = en_j, 1 \leq i \leq I, 1 \leq j \leq J\}$ as a chromosome. $s_i = en_j$ means that task $i$ is assigned to edge node $j$. We define the fitness function as follows.

$$Fitness(TK, EN, X) = \frac{1}{C(X)} \tag{15}$$

$$C(X) = \sum_{i,j}(x_{i,j} \times t_{i,j}^{ect}) + (\sum_{i} x_{i,j} - 1) \times T_j^{Avg} \tag{16}$$

Referring partly to the experiment of [7], we use the roulette-wheel method in the selection phase. In the crossover phase, we choose the standard one-point crossover operator and set the cross probability to 0.3. In the mutation phase, we choose the standard uniform mutation operator and set the mutation probability to 0.3. The population size is set to 20. We limit the maximum iteration number to 100, in order to ensure an negligible overhead of calculation compared with task completion time.

## 5 Experimental Evaluation

### 5.1 Experimental Setup

A prototype is implemented to evaluate RATA. Controller module runs on a desktop PC, while the edge node modules run on heterogeneous mobile devices such as laptops and Raspberry Pis. Modules are written in GoLang and launched in Linux environment, using gPRC for communication between modules. When a new edge node comes up, the edge node calls *JoinGroup* to register itself first, then starts calling *Heartbeat* to report its information periodically. When exiting, the edge node module calls *ExitGroup* to logout. The controller module calls *AssignTask* to forward tasks to edge node modules, which call *ReturnResult* to return results. To evaluate the intrinsic ability of reducing abortions, we don't apply any checkpointing techniques.

**Hardware Configuration:** Table 1 listed hardware characteristics of devices used for testbed experiment as edge nodes. Besides, a desktop PC equipped with an Intel(R) Core(TM) i5-8400 processor, 16GB RAM and Gigabit Network Adapter (connected to LAN via cable) is used as the Controller.

**Workload:** We use synthetic matrix multiplication Python tasks derived from the Google cluster trace [15], which is also used in experiments of previous works [12,14]. As tasks with short execution time are more suitable in edge computing with mobile devices, we filter out tasks whose execution time are larger than

**Table 1.** Hardware characteristics of edge node devices

| Device | Computation capacity | Bandwidth | Connection |
|---|---|---|---|
| 2 × Raspberry Pi 3B+ | 2.7 MFLOPs | 13.2 Mbps | Wireless LAN |
| 3 × Raspberry Pi 4B | 15.1 MFLOPs | 15.3 Mbps | Wireless LAN |
| Surface Pro 4(M3) | 8.7 MFLOPs | 95.9 Mbps | Wireless LAN |

10 min. We cluster the remaining tasks using k-means clustering on their execution time, and draw tasks from each task class proportionally to generate the workload. We bind each task in the workload with three features: user name of the task, task name, and logic task name. Since the tasks derived from the Google cluster trace don't have task names, we replace their job name by their task name, so do the logic task name. We generate the size of input data file of each task using a uniform distribution U(0MB, 10 MB).

In order to take into account the impact of deadline, we use deadline slack [14] to generate deadlines for 50% of the tasks in the workload randomly. Deadline slack is defined as follows.

$$deadlineslack = \frac{(deadline - executiontime)}{executiontime} \times 100\% \tag{17}$$

By default, we use a uniform distribution U(250%,300%) to generate deadline slacks, which are experimented in [18].

We totally generate 1200 tasks for testbed experiment and 15000 tasks for simulation experiments. For each experiment, the workload will be divided into 6 groups, we randomly choose a group to pre-train the Predictor, and the rest groups to conduct evaluation. We use a Poisson arrival process to model the arrival of tasks.

**Baseline:** We compare the performance of our risk-aware task assignment scheme (RATA) with the following existing task assignment schemes.

- **Shortest Completion Time First (SCTF):** Dispatch a task to the edge node with shortest completion time greedily.
- **Minimize Batch Completion Time (MBCT):** Use genetic algorithm to solve the original task assignment problem as formulation (2)–(4).
- **Femtocloud:** Femtocloud* [10] first sorts edge nodes by churn probability and gets a temporary best node. Then it iterates on the sorted list of edge nodes, comparing each candidate node with the best one in terms of gain (relative difference of completion time) and risk (relative difference of churn probability). If the gain exceeds the risk, it switches these two edge nodes.

**Metrics:** For tasks without deadlines, we use average completion time and average number of executions to measure the performance of task assignment. For tasks with deadlines, we use deadline miss rate to measure the performance. These metrics are listed below.

- **Average Completion Time:** average time tasks consume to get completely executed.
- **Average Number of Executions:** average times that tasks are executed (include abortions).
- **Deadline Miss Rate:** the percentage of tasks missed their deadline.

## 5.2 Testbed Experiment

In testbed experiments, similarly with Femtocloud [10], once an edge node leaves, it will return after an OFF period which follows a normal distribution with mean equals to 20% of the average presence time. Therefore, each edge node's duty cycle is set to 80% on average. Some parameters are shown in Table 2.

**Table 2.** Parameters of testbed experiment

| Parameter | | Value |
|---|---|---|
| Duration of each assignment cycle | | 1 s |
| Worker thread at each edge node | | 3 |
| Edge node queue management policy | | First-Come-First-Serve |
| Average presence time | Raspberry Pi 3B+ | 25 min |
| | Raspberry Pi 4B | 15 min |
| | Surface Pro 4(M3) | 20 min |

Figure 4 shows the performance of all these task assignment schemes on our testbed. Overall, RATA has the shortest average completion time and lowest deadline miss rate among listed schemes. Although RATA has a similar average number of executions with Femtocloud, the average completion time is shorter and deadline miss rate is lower. In simple terms, Femtocloud tends to select stabler edge nodes which may lead to longer task completion time to acquire lower abortion probability. As for MBCT, RATA outperforms it because the risk-aware mechanism reduces the abortions of task execution, which is reflected by the lower average number of executions. SCTF performs worst since it minimizes neither the batch completion time nor the abortion risk.

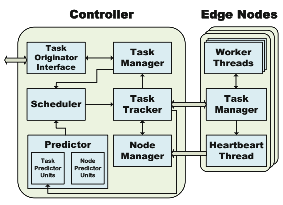


**Fig. 3.** System architecture.

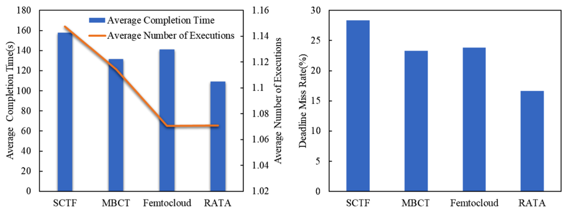


**Fig. 4.** Result of the testbed experiment.

## 5.3 Simulation Experiment

Simulation experiments are launched with larger number of simulated edge nodes. Each simulated edge node is an edge node module running on the desktop PC. We generate the bandwidth of simulated edge nodes using a uniform distribution U(8 Mbps, 80 Mbps). Transmission processes are simulated by a sleep function. A Poisson arrival process is used to model the arrival of edge nodes, and a normal distribution is used to generate presence time of each edge node. The standard deviation is set to 20% of the mean, similarly to Femtocloud. We study the impact of some parameters on task assignment performance in this experiment. Variations of these parameters can affect load status and the abortion risk. Default values of them are listed in Table 3.

**Table 3.** Default parameters of simulation experiments

| Parameter | Value |
|---|---|
| Task arrival rate | 180 tasks/min |
| Average presence time | 20 min |
| Edge node arrival rate | 5 nodes/min |

**Impact of Task Arrival Rate:** Task arrival rate directly affects the waiting queue length in edge nodes. As Fig. 5 shows, all metrics increase as this rate increases. However, RATA has the shortest average completion time and lowest deadline miss rate in most cases. Compared with MBCT, RATA has a similar average completion time but a lower deadline miss rate and average number of executions. Femtocloud has a similar average number of executions but a longer average completion time compared with RATA. Since RATA can quantify the expected overhead time brought by an abortion, it will compare a potential longer overhead time with a shorter task completion time when making task assignment decisions.

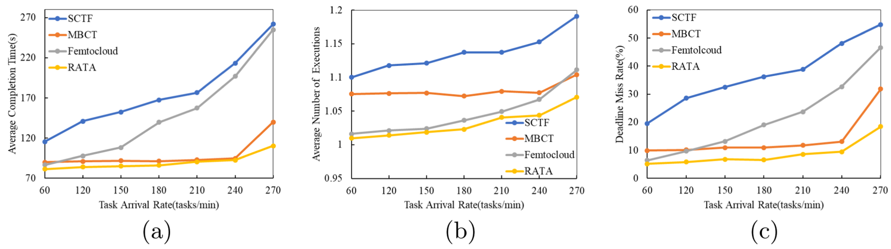


**Fig. 5.** Impact of task arrival rate. (a) Average completion time. (b) Average number of executions. (c) Deadline miss rate.

**Impact of Edge Node Arrival Rate:** Edge node arrival rate directly affects the number of edge nodes in the edge cloud. Figure 6 shows the impact of edge node arrival rate. Overall, all the metrics decrease as the arrival rate increases, and RATA outperforms other baselines in terms of three metrics in most cases. RATA has a similar performance with Femtocloud to reduce abortions of task execution in terms of average number of executions, but has a shorter average completion time and a lower deadline miss rate. Compared with MBCT, RATA has a similar performance on average completion time, but a lower average number of executions and deadline miss rate.

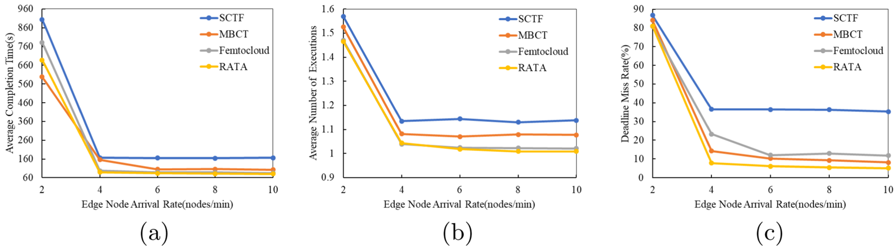


**Fig. 6.** Impact of edge node arrival rate. (a) Average completion time. (b) Average number of executions. (c) Deadline miss rate.

**Impact of Edge Node Heterogeneity:** This aims to test the ability to make full use of edge nodes with more computation capacity but shorter presence time. A new group of simulated edge nodes with this characteristic are newly introduced, modeled as a normal distribution N(5 min, 30 s). Figure 7 shows the impact of changing the arrival rate of these simulated edge nodes. Overall, all the metrics except the average number of executions decrease as the arrival rate increases. Femtocloud performs better than other schemes including RATA on keeping low average number of executions. This result shows the difference that RATA prefers to choose those edge nodes which can offer a shorter completion time for tasks despite of the higher abortion risk, while Femtocloud prefers stabler edge nodes.

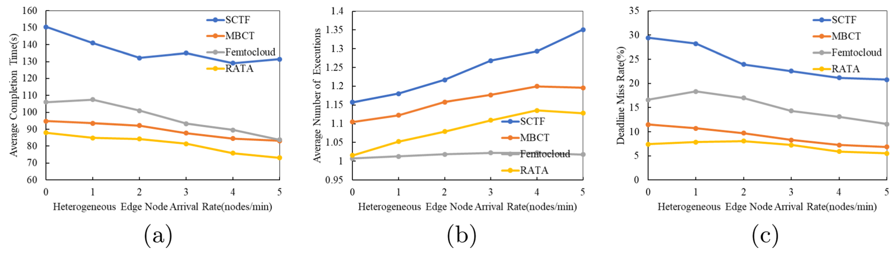
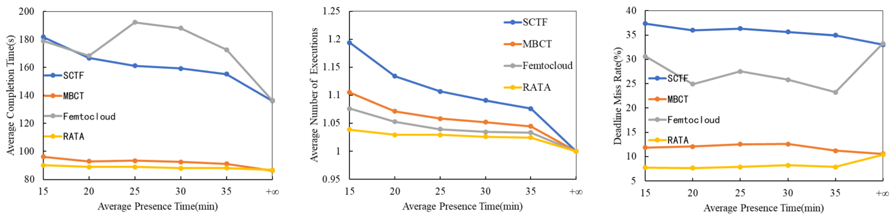


**Fig. 7.** Impact of edge node heterogeneity. (a) Average completion time. (b) Average number of executions. (c) Deadline miss rate.

**Impact of Average Presence Time of Edge Nodes:** This directly affects the abortion probability of task execution. Once an edge node leaves, it will return after an OFF period. The OFF period is modeled the same as the testbed experiment. Figure 8 shows the impact of changing the average presence time, where $+\infty$ means the edge nodes will never leave during the experiment. Overall, most metrics decrease as the presence time increases. RATA and MBCT have the shortest average completion time and lowest deadline miss rate, while RATA and Femtocloud outperform other baselines in terms of average number of executions.



(a) (b) (c)

**Fig. 8.** Impact of average presence time of edge nodes. (a) Average completion time. (b) Average number of executions. (c) Deadline miss rate.

## 6 Conclusion

In this work, we study the risk-aware optimization of task assignment in edge cloud consisting of dynamic mobile devices. We formulate a risk-aware task assignment problem, which aims to reduce the average completion time as well as abortions of task execution, and give a solution to this problem. We design and implement a prototype system to evaluate the method. Experiment results show that our scheme outperforms the state-of-the-art work in terms of average completion time and deadline miss rate in most cases.

**Acknowledgment.** This work is supported by National Natural Science Foundation of China (62172155, 62072465).

## References

1. Abbas, N., Zhang, Y., Taherkordi, A., Skeie, T.: Mobile edge computing: a survey. IEEE Internet Things J. **5**(1), 450–465 (2017)
2. Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: Fifth IEEE/ACM International Workshop on Grid Computing, pp. 4–10. IEEE (2004)
3. Bahreini, T., Grosu, D.: Efficient placement of multi-component applications in edge computing systems. In: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, pp. 1–11 (2017)

4. Ben-Haim, Y., Tom-Tov, E.: A streaming parallel decision tree algorithm. J. Mach. Learn. Res. **11**(Feb), 849–872 (2010)
5. Cao, G., Singhal, M.: Mutable checkpoints: a new checkpointing approach for mobile computing systems. IEEE Trans. Parallel Distrib. Syst. **12**(2), 157–172 (2001)
6. Chen, X., Lyu, M.R.: Performance and effectiveness analysis of checkpointing in mobile environments. In: 22nd International Symposium on Reliable Distributed Systems 2003, Proceedings, pp. 131–140. IEEE (2003)
7. Deng, S., Huang, L., Taheri, J., Zomaya, A.Y.: Computation offloading for service workflow in mobile cloud computing. IEEE Trans. Parallel Distrib. Syst. **26**(12), 3317–3329 (2014)
8. Guo, S., Chen, M., Liu, K., Liao, X., Xiao, B.: Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing. IEEE Trans. Mob. Comput. **20**(5), 2025–2040 (2020)
9. Habak, K., Ammar, M., Harras, K.A., Zegura, E.: Femto clouds: leveraging mobile devices to provide cloud service at the edge. In: 2015 IEEE 8th International Conference on Cloud Computing, pp. 9–16. IEEE (2015)
10. Habak, K., Zegura, E.W., Ammar, M., Harras, K.A.: Workload management for dynamic mobile device clusters in edge femtoclouds. In: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, pp. 1–14 (2017)
11. Liu, F., Guo, Y., Cai, Z., Xiao, N., Zhao, Z.: Edge-enabled disaster rescue: a case study of searching for missing people. ACM Trans. Intell. Syst. Technol. (TIST) **10**(6), 1–21 (2019)
12. Meng, J., Tan, H., Xu, C., Cao, W., Liu, L., Li, B.: Dedas: online task dispatching and scheduling with bandwidth constraint in edge computing. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 2287–2295. IEEE (2019)
13. Omara, F.A., Arafa, M.M.: Genetic algorithms for task scheduling problem. In: Abraham, A., Hassanien, AE., Siarry, P., Engelbrecht, A. (eds.) Foundations of Computational Intelligence Volume 3. Studies in Computational Intelligence, vol. 203, pp. 479–507. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01085-9_16
14. Park, J.W., Tumanov, A., Jiang, A., Kozuch, M.A., Ganger, G.R.: 3sigma: distribution-based cluster scheduling for runtime uncertainty. In: Proceedings of the Thirteenth EuroSys Conference, pp. 1–17 (2018)
15. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format+schema. Google Inc., White Paper, pp. 1–14 (2011)
16. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
17. Tumanov, A., Jiang, A., Park, J.W., Kozuch, M.A., Ganger, G.R.: JamaisVu: robust scheduling with auto-estimated job runtimes. Technical report CMU-PDL-16-104. Carnegie Mellon University (2016)
18. Tumanov, A., Zhu, T., Park, J.W., Kozuch, M.A., Harchol-Balter, M., Ganger, G.R.: TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: Proceedings of the Eleventh European Conference on Computer Systems, pp. 1–16 (2016)
19. Wu, H., et al.: Resolving multi-task competition for constrained resources in dispersed computing: a bilateral matching game. IEEE Internet Things J. **8**(23), 16972–16983 (2021)
20. Xu, Y., Li, K., Hu, J., Li, K.: A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf. Sci. **270**, 255–287 (2014)

21. Zhang, D., Ma, Y., Zhang, Y., Lin, S., Hu, X.S., Wang, D.: A real-time and non-cooperative task allocation framework for social sensing applications in edge computing systems. In: 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 316–326. IEEE (2018)
22. Zhang, D., Ma, Y., Zheng, C., Zhang, Y., Hu, X.S., Wang, D.: Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 243–259. IEEE (2018)
23. Zhang, D.Y., Wang, D.: An integrated top-down and bottom-up task allocation approach in social sensing based edge computing systems. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 766–774. IEEE (2019)