Cloud resource allocation for cloud-based automotive applications☆

Zhaojian Li[a], Tianshu Chu[b], Ilya V. Kolmanovsky[c], Xiang Yin[1,*,d], Xunyuan Yin[e]

[a] Department of Mechanical Engineering, Michigan State University, East Lansing, MI 48824, USA
[b] Department of Civil and Environmental Engineering, Stanford University, CA 94305, USA
[c] Department of Aerospace Engineering, The University of Michigan, Ann Arbor, MI 48109, USA
[d] Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China
[e] Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada

ARTICLE INFO

ABSTRACT

There is a rapidly growing interest in the use of cloud computing for automotive vehicles to facilitate computation and data intensive tasks. Efficient utilization of on-demand cloud resources holds a significant potential to improve future vehicle safety, comfort, and fuel economy. In the meanwhile, issues like cyber security and resource allocation pose great challenges. In this paper, we treat the resource allocation problem for cloud-based automotive systems. Both private and public cloud paradigms are considered where a private cloud provides an internal, company-owned internet service dedicated to its own vehicles while a public cloud serves all subscribed vehicles. This paper establishes comprehensive models of cloud resource provisioning for both private and public cloud-based automotive systems. Complications such as stochastic communication delays and task deadlines are explicitly considered. In particular, a centralized resource provisioning model is developed for private cloud and chance constrained optimization is exploited to utilize the cloud resources for best Quality of Services. On the other hand, a decentralized auction-based model is developed for public cloud and reinforcement learning is employed to obtain an optimal bidding policy for a "selfish" agent. Numerical examples are presented to illustrate the effectiveness of the developed techniques.

1. Introduction

There is growing interest in employing cloud computing in automotive applications [4,8,14,17,25,30–32]. Ready access to distributed information and computing resources can enable computation and data intensive vehicular applications for improved safety, drivability, fuel economy, and infotainment. Several cloud-based automotive applications have been identified. For instance, a cloud-based driving speed optimizer is studied in [22] to improve fuel economy for everyday driving. In [16], a cloud-aided comfort-based route planner is prototyped to improve driving comfort by considering both travel time and ride comfort in route planning. A cloud-based semi-active suspension control is studied in [15] to enhance suspension performance by utilizing road preview and powerful computation resources on the cloud.

As such, cloud computing has been both an immense opportunity and a crucial challenge for vehicular applications: opportunity because of the great potential to improve safety, comfort, and enjoyment; challenge because cyber-security and resource allocation are critical issues that need to be carefully considered. A cloud resource allocation scheme determines how a cloud server such as Amazon "EC2" or Google Cloud Platform distributes resources to its many clients (vehicles in our context) efficiently, effectively, and profitably. This allocation design becomes even more challenging when it comes to cloud-based automotive systems in which issues like communication delays and task deadlines arise. These complexities make a good resource allocation design a non-trivial, yet important task.

Not surprisingly, extensive studies have been dedicated to the development of efficient and profitable cloud resource allocation schemes. A dynamic bin packing method, MinTotal, is developed in [13] to minimize the total service cost. In [5], a distributed and hierarchical component placement algorithm is proposed for large-scale cloud systems. A series of game theoretical cloud resource allocation approaches have also been developed, see e.g., [2,3,11,19]. However, as far as the authors are aware, a resource allocation scheme for cloud-based automotive systems that accounts for communication delays and task deadlines is still lacking.

In this paper, we develop resource allocation schemes for cloud-based automotive systems that optimally tradeoff costs and Quality of Service

---

(QoS) with the presence of stochastic communication delays and task deadlines. In particular, we consider allocation schemes under two cloud paradigms, private and public cloud. A private cloud is a company-owned resource center which provides computation, storage and network communication services and is only accessible by cars made by the car company. The private cloud therefore has a high level of security and information is easy and safe to share and manage. On the other hand, a public cloud relies on a third-party service provider (e.g., Amazon EC2) that provides services to all subscribed vehicles. A public cloud can eliminate the capital expenses for infrastructure acquisition and maintenance, and can provide the service on an as-needed basis.

The objectives of resource allocation are quite different between private and public cloud paradigms. Since the private cloud resources are pre-acquired, the company basically "use them or waste them". Therefore, the goal of private cloud resource allocation is to best utilize its resources to provide good QoS to its subscribed vehicles. Since the information exchange between vehicles and the server is more secure and convenient, the resource allocation can be achieved in a centralized manner. On the other hand, public cloud provides services to subscribed vehicles from a variety of makers, e.g., Ford, GM, Toyota, etc. Due to security and privacy issues, these vehicles typically will not share their information nor be interested in coordination; hence each vehicle becomes a "selfish" agent. The goal of each agent is to minimize its service cost while maintaining good QoS.

In this work, we develop mathematical models to formalize the resource allocation problems for both private and public cloud paradigms. Stochastic communication delays and onboard task deadlines are explicitly considered. A centralized resource-provisioning scheme is developed for private cloud and chance constrained optimization is employed to obtain an optimal allocation strategy. On the other hand, an auction-based bidding framework is developed for public cloud and reinforcement learning is exploited to train an optimal bidding policy to minimize the cost while maintaining good QoS. Numerical examples are presented to demonstrate the effectiveness of the proposed schemes.

The main contributions of this paper include the following. Firstly, compared to the previous literature on cloud resource allocation, issues important to automotive vehicles such as communication delays and on-board task deadlines are explicitly treated in this paper. Secondly, resource allocation within a private cloud paradigm is formalized as a centralized resource partitioning problem. Chance constrained optimization techniques are employed to obtain the optimal partitioning by solving a convex optimization problem. Thirdly, a decentralized, auction-based bidding framework is developed for public cloud-based resource allocation and the best response dynamics assuming a constant time delay and bidding is derived. Furthermore, a Deep Deterministic Policy Gradient (DDPG) algorithm is exploited to train the optimal bidding policy with stochastic time delay and unknown bidding from other vehicles. Sensitivity analysis is also performed to show how the bidding policy can change by varying task parameters such as workload and deadline.

The rest of our paper is organized as follows. Section 2 describes the model of cloud resource provisioning for private cloud-based automotive systems. The problem formulation and a chance constrained optimization approach are also presented. In Section 3, a numerical example is given to illustrate the allocation scheme for private cloud. The resource allocation problem with a public cloud is formalized in Section 4. The best response dynamics with constant time delay and bidding is also derived. A DDPG algorithm is exploited in Section 5 to train the optimal bidding policy with stochastic time delay and unknown bidding from other vehicles. A numerical case study is also presented with sensitivity analysis on task parameters. Finally, conclusions are drawn in Section 6.

2. Centralized resource allocation with a private cloud

It is more secure and manageable for automotive manufacturers to acquire and maintain its own private cloud infrastructure which provides computation, data storage and network services only to vehicles
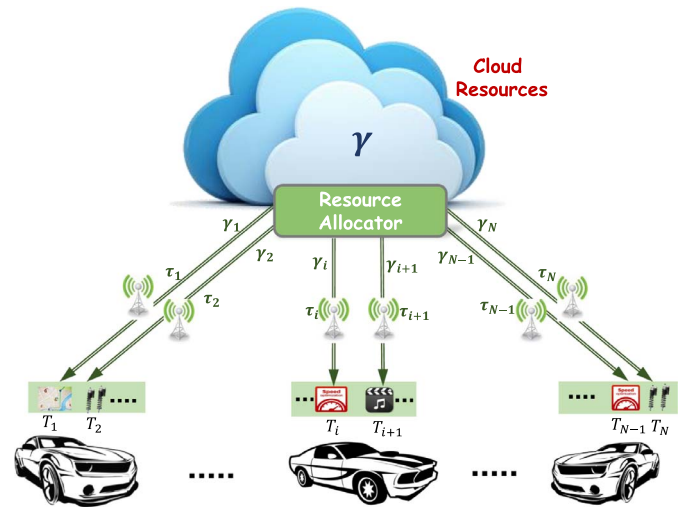

Fig. 1. Schematic diagram of private cloud-based resource allocation.

made by the manufacturer. A schematic diagram of resource allocation for private cloud-based automotive systems is illustrated in Fig. 1. Suppose that a set of cloud-based vehicular applications are available (e.g., cloud-based route planning, cloud-based suspension control, etc.) and we consider a general case that each vehicle runs a subset of these applications. Let us consider a total number of N applications running on M vehicles as in Fig. 1. Each application i, $i = 1, 2, ..., N$, corresponds to a periodic task associated with a tuple, $\mathcal{T}_i = \{T_i, w_i, d_i, \tau_i\}$, where

- T_i is the period of task i in seconds;
- w_i is the workload of task i in million instructions;
- $d_i \leq T_i$ is the deadline of task i in seconds;
- τ_i is a random time delay of the communication channel associated with task i in seconds.

For each task i, the Quality of Service (QoS) is characterized by the following cost function adopted from [33]:

$$C_i(\gamma_i; \tau_i) = \begin{cases} B_i\left(\frac{w_i}{\gamma_i} + \tau_i\right), & \text{if } \frac{w_i}{\gamma_i} + \tau_i \leq d_i \\ M_i, & \text{Otherwise}, \end{cases} \tag{1}$$

where γ_i is the process rate that the cloud resource allocator assigns to task i and $\sum_{i=1}^{N} \gamma_i = \gamma$ with γ being the total resource available on the cloud; $B_i(\cdot): \mathbb{R}^+ \to \mathbb{R}^+$ is a non-decreasing function reflecting the QoS of task i; $M_i \geq B_i(d_i)$ is a positive scalar representing the penalty for missing the deadline; the condition $\frac{w_i}{\gamma_i} + \tau_i > d_i$ indicates that the deadline has been missed. Note that task priorities are reflected in the deadline-missing penalty M_i. For safety-critical tasks (e.g., cloud-based functions involved in powertrain or vehicle control), a large penalty, M, should be given while a small M can be assigned to some non-critical tasks such as online video streaming.

Since a private cloud is a pre-acquired "use it or waste it" capability, the goal of resource allocation for private cloud-based automotive systems is to distribute the cloud resources to the N tasks such that the total expected QoS cost as in (1) is minimized. Basically, the cloud collects task information (i.e., workload, deadline, time delay statistics[2]) of the N tasks and determines how optimally to partition the total resources into N parts so that the expected QoS cost is minimized. The problem can be mathematically formalized as a constrained optimization problem

---

[2] Note that the task period T_i is not used here but we include it as one of the four task attributes for completeness. The task period will appear when it comes to the public cloud-based resource allocation in Section 4.

$$\min_{\Gamma} J(\Gamma) = \mathbb{E}\left[\sum_{i=1}^{N} C_i(\gamma_i; \tau_i)\right]$$

$$\text{Subject to:} \quad \sum_{i=1}^{N} \gamma_i = \gamma,$$

$$\gamma_i \geq 0, \ \forall \ i = 1, \cdots, N \quad (2)$$

where $\Gamma = [\gamma_1, \gamma_2, \cdots, \gamma_N]^{\mathrm{T}}$ is the vector of process rates to be optimized.

We note that the problem (2) is challenging to solve due to the randomness of communication delay τ_i and the discontinuity of the cost function represented in (1). Motivated by the chance constrained formulation in Stochastic Model Predictive Control developments [7,21,23], we re-formulate problem (2) by imposing chance constraints. Instead of penalty for missing deadline as in (1), we impose chance constraints for missing deadlines of the form,

$$\Pr\left(\frac{w_i}{\gamma_i} + \tau_i \leq d_i\right) \geq 1 - \alpha_i, \quad (3)$$

where $\alpha_i \in (0, 1)$ is a scalar representing the chance constraint of missing a deadline, $i = 1, \cdots, N$. The notion of α can be interpreted as the upper limit of deadline missing rate specified in the QoS requirements. Applications with harsh consequences of missing a deadline can be characterized by a small α while larger α can be used for applications with mild consequences of missing a deadline.

Note that the deadline-missing penalty M and the chance constraint α are transformable. For instance, one can use the following function to map deadline-missing penalties to chance constraints:

$$\alpha_i = \alpha_{\max} + \frac{\alpha_{\min} - \alpha_{\max}}{M_{\max} - M_{\min}}(M_i - M_{\min}), \quad (4)$$

where $\alpha_{\min}$ and $\alpha_{\max}$ are, respectively, the lower and upper bounds of the chance constraints while $M_{\min}$ and $M_{\max}$ are the corresponding lower and upper bounds of the deadline-violation penalties, respectively. These parameters need to be chosen compatibly to reflect the same QoS requirements. The example transformation in (4) is illustrated in Fig. 2. Now let us assume that the communication delays can be modeled as independent Gaussian random variables, i.e., $\tau_i \sim \mathcal{N}(\overline{\tau_i}, \sigma_i^2)$. From basic probability theory, the probability of the delay taking values between a and b,

$$\Pr(a < \tau_i \leq b) = \frac{1}{2}\mathrm{erf}\left(\frac{b - \overline{\tau_i}}{\sqrt{2}\sigma_i}\right) - \frac{1}{2}\mathrm{erf}\left(\frac{a - \overline{\tau_i}}{\sqrt{2}\sigma_i}\right), \quad (5)$$

where $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t$ is the error function.

As a result, from (3) and (5), it follows that

$$\Pr\left(\tau_i \leq d_i - \frac{w_i}{\gamma_i}\right) = \frac{1}{2}\mathrm{erf}\left(\frac{d_i - \frac{w_i}{\gamma_i} - \overline{\tau_i}}{\sqrt{2}\sigma_i}\right) + \frac{1}{2} \geq 1 - \alpha_i. \quad (6)$$

We next apply the inverse error function $\mathrm{erf}^{-1}(\cdot)$ to both sides of (6). Since $\mathrm{erf}^{-1}(\cdot)$ is continuous and increasing, we have

$$d_i - \frac{w_i}{\gamma_i} - \overline{\tau_i} \geq \sqrt{2}\sigma_i\mathrm{erf}^{-1}(1 - 2\alpha_i). \quad (7)$$

Note that (7) requires the term $d_i - \overline{\tau_i} - \sqrt{2}\sigma_i\mathrm{erf}^{-1}(1 - 2\alpha_i)$ to be positive so that γ_i is feasible. This condition means that the mean of the delay $\overline{\tau_i}$ cannot be greater than the deadline d_i. Also, given deadline d_i, delay mean $\overline{\tau_i}$ and standard deviation σ_i, the minimum achievable chance constraint level, α^* is

$$\alpha^* = \frac{1}{2} - \frac{\mathrm{erf}(d_i - \overline{\tau_i})}{2\sqrt{2}\sigma_i}, \quad (8)$$

which defines a maximum performance bound regardless of allocated resources. For example, if $d_i = 0.3$, $\overline{\tau_i} = 0.2$, and $\sigma_i = 0.1$, then from (8) we have $\alpha^* = 0.3976$, which means that no matter how many resources are allocated for task i, the probability of missing a deadline is no less than $\alpha^* = 0.3976$ due to communication delays. On the other hand,
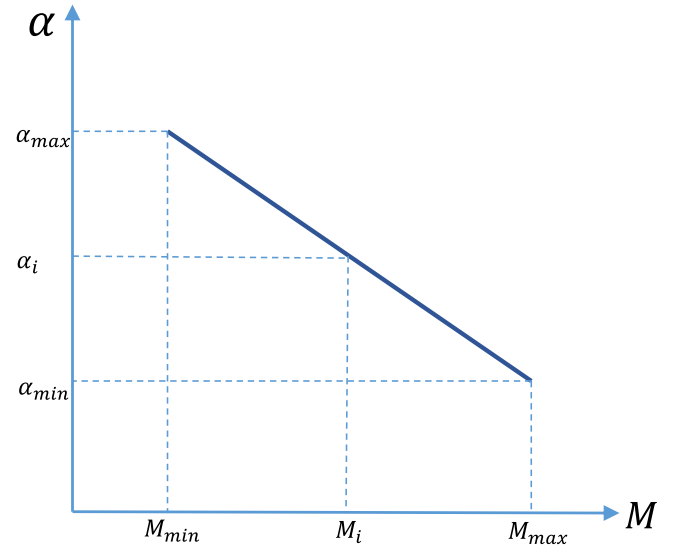

Fig. 2. Linear mapping from deadline-missing penalty to chance constraint.

$\alpha^* < 0$ means that the probability of not missing deadline can be infinitely close to 1 with enough resources.

Re-arranging terms in (7) leads to

$$\gamma_i \geq \rho_i = \frac{w_i}{d_i - \overline{\tau_i} - \sqrt{2}\sigma_i\mathrm{erf}^{-1}(1 - 2\alpha_i)}. \quad (9)$$

So far we showed that (9) and (3) are equivalent. Therefore, the problem in (2) can be re-stated as:

$$\min_{\Gamma} J(\Gamma) = \mathbb{E}\left[\sum_{i=1}^{N} B_i\left(\frac{w_i}{\gamma_i} + \tau_i\right)\right]$$

subject to:

$$\sum_{i=1}^{N} \gamma_i = \gamma,$$

$$\gamma_i \geq \rho_i > 0, \quad \forall \ i = 1, 2, ..., N, \quad (10)$$

where ρ_i are assumed to be positive and defined by (9).

Note that if we choose $B(\cdot)$ to be a convex function of γ_i, as we will show in the next section, problem (10) reduces to a convex optimization problem which can be efficiently solved with good scalability [9].

3. A numerical example with a linear QoS function

In this section, we consider a linear QoS function in the form of $B_i\left(\frac{w_i}{\gamma_i} + \tau_i\right) = b_i \cdot \left(\frac{w_i}{\gamma_i} + \tau_i\right)$ with $b_i > 0$. The problem (10) becomes

$$\min_{\Gamma} J(\Gamma) = \sum_{i=1}^{N} b_i \cdot \left(\frac{w_i}{\gamma_i} + \overline{\tau_i}\right)$$

subject to:

$$\sum_{i=1}^{N} \gamma_i = \gamma,$$

$$\gamma_i \geq \rho_i > 0, \quad \forall \ i = 1, 2, ..., N. \quad (11)$$

We show that the above problem is a convex optimization problem. We first demonstrate that the cost function $J(\Gamma)$ in (11) is strictly convex in the domain $\{\Gamma = [\gamma_1, \cdots, \gamma_N]^{\mathrm{T}} : \gamma_i > 0, \forall \ i = 1, 2, \cdots N\}$. Towards that end, we compute the Hessian of the cost function J as

$$H_{xx}(J(\Gamma)) = diag\left\{\frac{2b_1 w_1}{\gamma_1^3}, \cdots, \frac{2b_N w_N}{\gamma_N^3}\right\}, \quad (12)$$

where $H_{xx}(\cdot)$ represents the Hessian matrix and $diag\{\cdot\}$ denotes the diagonal matrix with the arguments as the entries in the diagonal. Since

Table 1
Parameters for numerical example.

Attribute\Task	One	Two	Three	Four
Workload (w_i, in Million instructions)	0.02	0.03	0.1	0.12
Deadline (d_i, in seconds)	0.25	0.35	0.4	0.6
QoS cost scalar (b_i, in $/s$)	1	2	2	3
Delay mean ($\bar{\tau}_i$, in seconds)	0.1	0.1	0.08	0.11
Delay standard deviation (σ_i, in second)	0.02	0.03	0.02	0.03
Chance limit (α_i^* from (8))	−2.47	−2.76	−5.67	−5.53
Chance constraint (α_i, unitless)	0.1	0.05	0.02	0.01

b_i, w_i, γ_i are positive for $i = 1, 2, \cdots, N$, we have $H_{xx}(J(\Gamma))$ being positive definite, which means that the cost function $J(\Gamma)$ is strictly convex. Furthermore, the constraints in (11) are polytopic. Therefore, (11) is a convex optimization problem that can be efficiently solved by many numerical solvers. This means even if N is large, an optimal resource allocation can be efficiently computed.

We next give a numerical example with four tasks. The parameters are given in Table 1 and we consider a total resource of 1, i.e., $\gamma = 1$. The *fmincon* function in MATLAB was exploited to solve (11) and the optimized allocation strategy is,

$$\gamma_1 = 0.1608, \quad \gamma_2 = 0.1495, \quad \gamma_3 = 0.3585, \quad \gamma_4 = 0.3312. \tag{13}$$

To verify the chance constraints with the optimized allocation scheme, we run simulations under the allocation policy (13) for 10^6 times with the random delays specified in Table 1. The deadline violation rates for the four applications are, respectively, 0.0961, 0.0482, 0.01991, and 1.3×10^{-5}, which are all smaller than the specified chance constraints in Table 1. This means that the specified chance constraints are satisfied under the allocation scheme (13).

4. Decentralized resource allocation for public cloud paradigm

4.1. Problem formulation

An automotive manufacturer may choose to subscribe its cloud-based automotive applications to a public cloud without acquiring and maintaining its own infrastructure. A public cloud such as Amazon EC2 offers an on-demand and pay-as-you-go access over a shared pool of computation resources. This public cloud provides services to a large number of automobiles from a variety of manufacturers. These vehicles may not want to share either their resource policies or task information with other vehicles, which makes it impossible to run a centralized allocation scheme as in the private cloud paradigm. Instead, each vehicle becomes a "selfish" client that seeks to minimize its own cost while maintaining good QoS.

We consider a decentralized auction-based resource allocation model as illustrated in Fig. 3. For the considered vehicle, let N denote the number of tasks that are running in the vehicle and each task is associated with the same tuple $\mathcal{T}_i = \{T_i, w_i, d_i, \tau_i\}$ as defined in Section 2. The public cloud is running an auction-based resource allocation scheme, that is, each vehicular task i, $i = 1, \cdots, N$, submits a bid p_i (in US dollars per second) and obtains a proportion of the total cloud resources as:

$$\gamma_i = \frac{p_i}{P} \cdot \gamma = \frac{p_i}{P^- + \sum_{i=1}^{N} p_i} \cdot \gamma, \tag{14}$$

where P is the sum of all bids the cloud receives from all vehicles; $P^- = P - \sum_{i=1}^{N} p_i$ is the cumulative bid from all *other* vehicles; and γ quantifies the total resources available on the cloud.

Since there are many other vehicles subscribed to the public cloud, it is reasonable to assume that $P^- >> \sum_{i=1}^{N} p_i$. From (14) it follows that

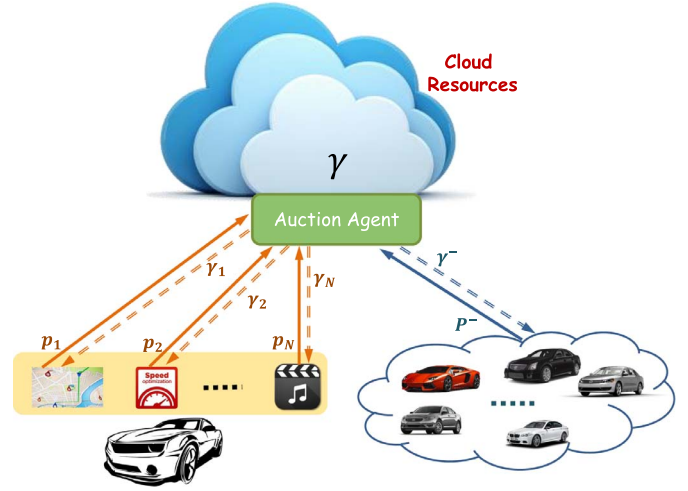$$\gamma_i \approx \frac{p_i}{P^-} \gamma, \tag{15}$$

Fig. 3. Schematic diagram of public cloud resource allocation .

which implies that the bidding policy of the tasks can be considered independently.

We consider a general bidding model in which the time period between the beginning of a period and the deadline is composed of multiple bidding steps. As illustrated in Fig. 4, there are l, $l \geq 1$, bidding steps before the deadline in each task period. With the QoS cost modeled in (1), the overall cost of task i in its period T_i is,

$$J_i = \sum_{t=1}^{l} p_{i,t} \cdot t_s + C_i(\gamma_i; \tau_i), \tag{16}$$

where $p_{i,t}$ is the bidding for task i at bidding step t; $t_s = \frac{d}{l}$ is the bidding time interval; and $C_i(\gamma_i; \tau_i)$ is the QoS cost defined in (1). The goal of the vehicle is to find an optimal bidding policy to minimize the accumulated cost (16) for each task. We next derive the optimal bidding strategy with a preliminary assumption that P^- and τ are known and constant. In Section 5, this assumption is removed.

4.2. Best response dynamics with constant P^- and τ

In this subsection, we seek to find the optimal bidding if bids from other vehicles (P^-) and the communication delays (τ) are known and constant. Since P^- is constant, all it matters is the total bidding. For task i, the optimal average bidding in the interval $[0, d_i - \tau_i]$ is defined as

$$
\begin{aligned}
p_i^* &= \arg\min_{p_i} J_i(p_i) \\
&\triangleq \arg\min_{p_i} p_i \cdot (d_i - \tau_i) + C_i(p_i; \tau),
\end{aligned} \tag{17}
$$

where $C_i(p_i; \tau)$ is defined in (1) and from (15) it follows that

$$C_i(p_i; \tau_i) = \begin{cases} B_i \left(\frac{w_i P^-}{p_i \gamma} + \tau_i \right), & \text{if } p_i \geq \frac{w_i P^-}{(d_i - \tau_i)\gamma} \\ M_i, & \text{Otherwise.} \end{cases} \tag{18}$$
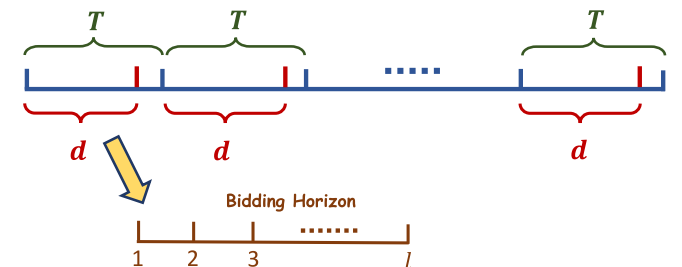

Fig. 4. A general bidding model with l bidding steps in one task period.

As a result, the overall cost function $J(p_i)$ becomes

$$J_i(p_i) = \begin{cases} p_i \cdot (d_i - \tau_i) + B_i\left(\frac{w_i P^-}{p_i \gamma} + \tau_i\right), & \text{if } p_i \geq \frac{w_i P^-}{(d_i - \tau_i)\gamma} \\ p_i \cdot (d_i - \tau_i) + M_i, & \text{Otherwise.} \end{cases} \tag{19}$$

Consider the linear QoS function $B_i(x) = b_i \cdot x$ with $b_i > 0$ as in Section 3. Then there are two local minimizers in (19): one associated with no bidding ($p_i^* = 0$), the other corresponds to the optimal bidding with no deadline missing. The second minimizer can be represented as

$$p_i^* = \max\left\{\sqrt{\frac{b_i w_i P^-}{(d_i - \tau_i)\gamma}}, \frac{w_i P^-}{(d_i - \tau_i)\gamma}\right\}, \tag{20}$$

depending whether the minimizer of the function $p_i \cdot (d_i - \tau_i) + B_i\left(\frac{w_i P^-}{p_i \gamma} + \tau_i\right)$, i.e., $\sqrt{\frac{b_i w_i P^-}{\gamma \cdot (d_i - \tau_i)}}$, can avoid deadline missing. An example of the cost function (19) with $w_i = 0.06$, $d = 0.4$, $\tau = 0.1$, $b_i = 8$, $P^- = 20$, $M_i = 5$, $t_s = 0.05$, and $\gamma = 10$ is illustrated in Fig. 5. The global minimizer is $\sqrt{\frac{b_i w_i P^-}{\gamma \cdot t_s}} = 1.7889$.

The assumption that the P^- and τ are constant and known is unrealistic in many applications. We next exploit a reinforcement learning framework to obtain the optimal bidding policy with no assumptions on the prior knowledge of P^- and τ.

5. Training optimal bidding policy using RL

5.1. Introduction to RL

Reinforcement learning (RL) is a data-driven approach for adaptively evolving optimal control policies based on the real-time measurements. Unlike traditional methods, RL models the stochastic ambiguity within the framework of Markov decision processes (MDP) [6], and learns the policy according to transition data observations [26]. There are commonly three types of RL algorithms: Q-learning, policy gradient, and actor-critic. The Q-learning (or approximate Q-learning) is the traditional RL algorithm that learns a Q-function $Q_\theta(s, a)$ with model parameter θ to estimate the delayed total reward of the current state and action a, and performs control as $\hat{a} \in \text{argmax}_a Q_\theta(s, a)$ [28] based on the learned policy. The Q-learning updates the Q-function parameters based on each observed temporal difference using stochastic gradient descent:

$$\theta \leftarrow \theta + \eta \Delta_t \phi(s_t, a_t), \tag{21}$$

where η is the learning rate, $\phi(s_t, a_t)$ is the input feature vector for the learning model, and Δ_t is the sampled temporal difference with stage reward r_t and discount factor α:
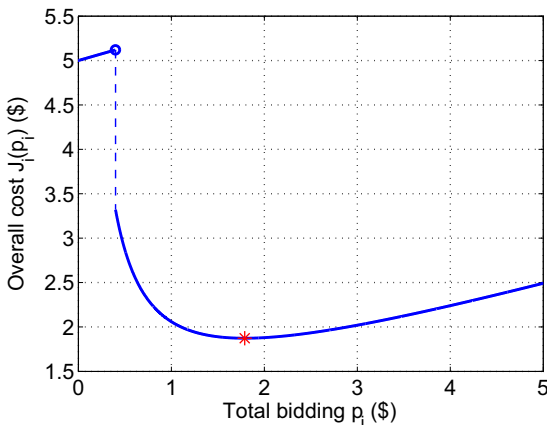

Fig. 5. Overall cost as a function of total bidding with $w_i = 0.06$, $d = 0.4$, $\tau = 0.1$, $b_i = 8$, $P^- = 20$, $M_i = 5$, $t_s = 0.05$, and $\gamma = 10$.

$$\Delta_t = r_t + \alpha \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t). \tag{22}$$

In the policy gradient approach, the stochastic optimal action distribution $\pi_\theta(a|s)$ with model parameter θ is learned directly, and control action is determined as $\hat{a} \in \text{argmax}_a \pi_\theta(a|s)$ [27]. Policy gradient updates the policy distribution based on each observed advantage function:

$$\theta \leftarrow \theta + \eta \nabla_\theta \log \pi_\theta(a_t|s_t)(\hat{Q}^\pi(s_t, a_t) - \hat{V}(s_t)), \tag{23}$$

where $\hat{Q}^\pi(s_t, a_t)$ is the sampled Q-value of (s_t, a_t) by following policy π_θ and $\hat{V}(s_t)$ is the sampled optimal value of s_t.

The actor-critic approach can be regarded as a combination of both since it learns both the policy $\pi_\theta(a|s)$ and the corresponding Q-function of the policy $Q_\beta^\pi(s, a)$ [12]. The details of the actor-critic updates will be covered in the following subsection.

All the above algorithms typically assume the discrete action space which, in particular, simplifies the search for $\hat{a}$. However, in our resource allocation problem, it is more natural to consider a continuous action space since the bid should be a continuous numerical number. For this case, deterministic policy gradient algorithm was developed recently that allows to directly learn the policy $\mu(s)$ instead of the policy distribution $\pi(a|s)$, and the control is simply performed as $\hat{a} = \mu(s)$ [24]. Then instead of the traditional ϵ−greedy exploration or Boltzmann exploration for Q-learning, we need to perform Ornstein–Uhlenbeck noise [29] to explore with the deterministic continuous policy.

In the following sections, we first formulate the bidding-based resource allocation problem. We then propose the corresponding MDP formulation for this stochastic optimal control problem. Further, we implement a deep network based actor-critic algorithm to learn the optimal bidding strategy using deterministic policy gradient. Finally, we evaluate the performance of this algorithm using various numerical experiments.

5.2. Training optimal bidding policy with deep deterministic policy gradient

In this section, we exploit RL to seek the optimal bidding policy. Towards that end, we first model the bidding process as a MDP, $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, r, \alpha\}$, where

- $\mathcal{S} = \{w_t, \Delta w_t, a_{t-1}, d_t\}$ represents the state space, where w_t is the remaining workload at time t; $\Delta w_t = w_{t-1} - w_t$ is the recent processed work; a_{t-1} represents the last bid, and d_t represents the remaining time until deadline. At the beginning of each period, the initial state is simply $s_0 = [w, 0, 0, d]$;
- $\mathcal{A} \in [0, +\infty)$ is the action space representing the bidding for the task;
- $\mathcal{P}$ is the transition matrix where each element $\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$ is the probability that the system transfers to s' from s given the current action a.
- r is a stage reward function given after each bidding to guide the optimal decision-making: $r(s, a) = \mathbb{E}[r_t|S_t = s, A_t = a]$.
- $\alpha \in [0, 1)$ is a discount factor that differentiates the importance of future rewards and present rewards.

Since the bidding of other vehicles is unknown so the transition matrix $\mathcal{P}$, traditional MDP optimizers such as Policy iteration and Value iteration cannot be directly applied. In this study, we exploit RL to learn an optimal bidding policy for each application. Furthermore, since the action space $\mathcal{A} = [0, +\infty)$ is continuous, approximate Q-learning and stochastic policy gradient algorithms cannot be applied without action discretization. To resolve this difficulty, we exploit the deterministic actor-critic (DAC) algorithm. In particular, we learn both a parameterized deterministic policy function $\mu_\theta: \mathcal{S} \rightarrow \mathcal{A}$ to perform the bidding action, and another parameterized critic Q-function $Q_\beta: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ to evaluate the bidding strategy. The bidding and learning procedure with a typical DAC algorithm is:

Parameters: $\alpha, \eta_\beta, \eta_\theta, l, K, W, w, d, \gamma, \delta$.

1 **initialize** networks $\theta = 0, \beta = 0$, target networks $\theta' = \theta, \beta' = \beta$;

2 **initialize** replay buffer $\mathcal{D}(K) = 0$;

3 **for** *each episode* **do**

4   **initialize** $s_0 = [w, 0, 0, d]$; **for** $t = 0 \rightarrow l - 1$ **do**

        `/* exploring step */`

5   **perform** action $a_t = \mu_\theta(s_t) + \epsilon_{OU}$;

        `/* observing step */`

6   **sample** γ_t from the environment;

7   **observe** s_{t+1}, r_t based on Eq. (15), Eq. (24) and Eq. (25);

8   **store** (s_t, a_t, s_{t+1}, r_t) in $\mathcal{D}$;

        `/* learning step */`

9   **sample** minibatch $\{(s_j, a_j, s_{j+1}, r_j)\}_{j=1}^{W}$ from $\mathcal{D}$;

10   **estimate** temporal difference $\Delta_t = r_t + \alpha Q_\beta(s_{t+1}, \mu_{\theta'}(s_{t+1})) - Q_\beta(s_t, a_t)$;

11   **update** networks β and θ with Eq. (27) and Eq. (28);

12   **update** target networks $\beta' \leftarrow \delta\beta + (1 - \delta)\beta'$, $\theta' \leftarrow \delta\theta + (1 - \delta)\theta'$;

13   **end**

14 **end**

15 **return** β, θ

Algorithm 1. Online minibatch DDPG training.

Table 2
Parameters of vehicular applications for simulation.

Attribute\Application	One	Two	Three	Four
Workload (w, in million instructions)	0.02	0.06	0.1	0.12
Deadline (d, in seconds)	0.5	0.4	0.4	0.6
Penalty for missing deadline (M, in $)	2	2	10	10

Table 3
Bidding policies for vehicular applications with fixed environment.

Variable\Application	One	Two	Three	Four
Best episode	2200	4700	700	4000
Best average total reward	−0.96	−2.00	−3.94	−4.75
Total bid (in $)	0.72	0.00	3.39	4.36
Equivalent bidding rate (in $/second)	1.80	0	11.30	8.72
Optimal bidding rate (in $/second)	1.83	0	11.00	7.92
Assigned workload (in million instructions)	0.02	0.00	0.10	0.13
Completion time (in seconds)	0.05	NA	0.15	0.20

Fig. 6. Three sampled P^- bidding trajectories from Ornstein–Uhlenbeck process.

1. At each time step $t \leq l$, observe the state s_t.
2. Perform a bid $\hat{a}_t$ based on the actor policy plus some random exploration perturbations, i.e., $\hat{a}_t = \mu_\theta(s_t)$+perturbations.
3. Receive the cloud resource allocation γ_t, and update the states as

$$\Delta w_{t+1} = \gamma_t \cdot t_s, \quad w_{t+1} = w_t - \Delta w_{t+1}, \quad d_{t+1} = d_t - t_s. \quad (24)$$

4. Terminate whenever the procedure is completed: $w_{t+1} \leq 0, d_{t+1} \geq 0$, or aborted: $w_{t+1} > 0, d_{t+1} = 0$.
5. Receive the current reward. If the procedure is aborted, receive a deadline-missing penalty $r_t = -M$; if the procedure is completed, a cost $r_t = -b \cdot t$ with b be a positive scalar representing the QoS cost coefficient is received; otherwise the agent receives the following state stage cost
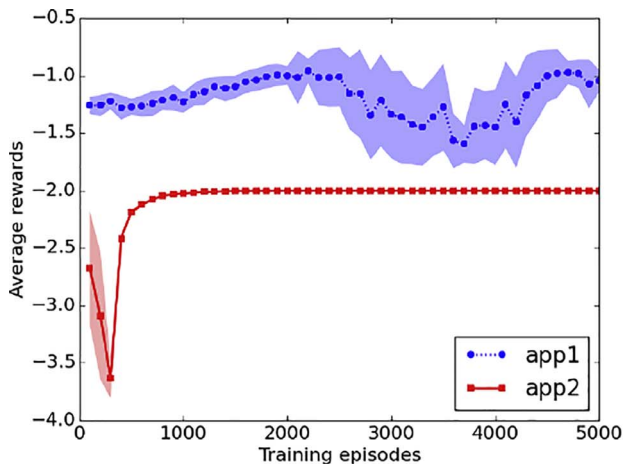
$$r_t = -\hat{a}_t \cdot t_s. \quad (25)$$

In Step 2, instead of performing an ε−greedy exploration over the entire action space, we add some Ornstein–Uhlenbeck noises into $\hat{a}_t$ to explore actions in the vicinity. A replay buffer is employed to store recent transitions (s_t, a_t, s_{t+1}, r_t) so that random transitions can be sampled to train the parameterized models to reduce the effects of data correlation [20]. When the replay buffer is filled, we can update both β and θ in the models by exploiting W randomly selected transitions from the buffer. The update of β is similar to the one in Q-learning: First we estimate the temporal difference from each selected transition:

$$\Delta_t = r_t + \alpha Q_\beta(s_{t+1}, \mu_\theta(s_{t+1})) - Q_\beta(s_t, a_t), \quad (26)$$

where $\alpha \in [0, 1)$ is the discount factor. We then update the parameter in the critic Q-function using stochastic gradient descent, i.e.,

$$\beta \leftarrow \beta + \frac{\eta_\beta}{W} \sum_{t=1}^{W} \Delta_t \nabla_\beta Q_\beta(s_t, a_t). \quad (27)$$

Finally, we update the parameter of the actor policy function based on the Q-function estimation

$$\theta \leftarrow \theta + \frac{\eta_\theta}{W} \sum_{t=1}^{W} \nabla_\theta \mu_\theta(s_t) \nabla_a Q_\beta(s_t, a) \bigg|_{a=\mu_\theta(s)}. \quad (28)$$

Here η_β and η_θ are positive constants representing the learning rates. In this study, we apply deep neural networks as the approximation functions for both the actor and critic. This specific implementation of the deterministic actor-critic (DAC) algorithm is referred to as the deep deterministic policy gradient (DDPG) [18]. Furthermore, techniques such as experience replay [20] and batch normalization [10] are also employed to improve the learning performance.

The complete DDPG algorithm is shown in Algorithm 1. The algorithm parameters include: the discount factor α, learning rates η_β, and η_θ in (26), (27), and (28), respectively; bidding horizon l as in Fig. 4; replay buffer size K; mini-batch size W, $W < K$; task workload w; task deadline d; total cloud resource γ, and parameter smoothing scalar $\delta \in (0, 1)$. Specifically, Line 1 initializes the network parameters and the
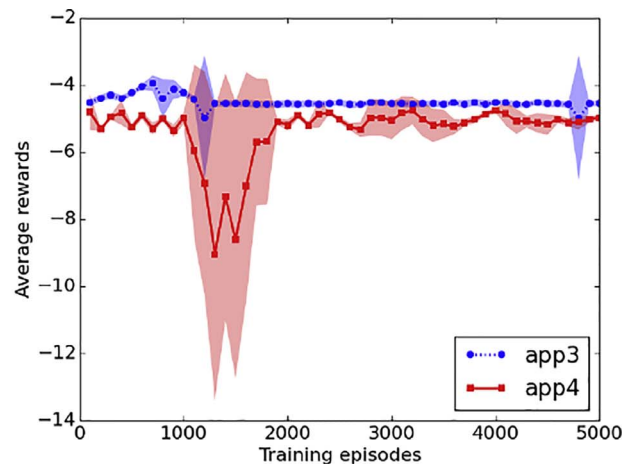

Fig. 7. Total rewards vs. training episodes. Left: application 1 and 2, right: application 3 and 4.
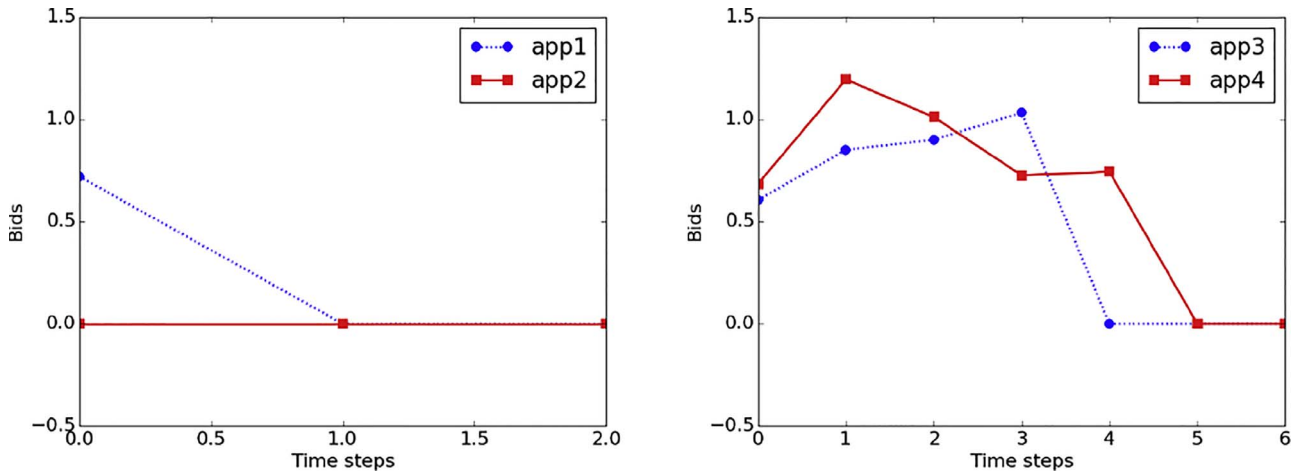
Fig. 8. Trained bidding vs. time steps. Left: application 1 and 2, right: application 3 and 4.

target network parameters for smooth updating. Line 2 initializes an experience replay $\mathscr{D}$ that stores the K most recent transition samples. At the beginning of each training episode (periodic task), we reset the state and sample time delay. At each time step, Line 6 performs exploration with some sampled Ornstein–Uhlenbeck noise ϵ_{OU}; Line 7 samples P^- from the environment; Lines 8–9 observe the system transition and add the current transition sample to the replay buffer; Lines 10–12 update the networks based on the sampled minibatch from the experience replay; Line 13 updates the corresponding target networks with a weighted sum to smooth the training.

5.3. Numerical experiments

5.3.1. Simulation setup

In this subsection, we perform simulations to illustrate the DDPG approach in Algorithm 1. Four tasks are considered in the host vehicle. The task specifications are listed in Table 2. The bidding policy of each application is trained separately. We define the total cloud resource as $\gamma = 1$ million instructions/second. The time delays of all applications are assumed to be the same and are $\tau \sim |\mathcal{N}(0.1, 0.0025)|$ in seconds. The bidding period t_s is set to 0.05 s so $20d$ gives bidding horizon l in steps and 20τ gives the delay in steps. We sample P^- from an Ornstein–Uhlenbeck process with $\mu = 33$, $\theta = 1$, $\sigma = 1.5$ \$/s to reflect similar prices from Amazon EC2 [1]. Three sampled trajectories of the Ornstein–Uhlenbeck process are illustrated in Fig. 6. We set $b = 2$ in the cost functions for all tasks.

For DDPG training, we train 5000 task periods with $\alpha = 0.99$, $\delta = 0.001$, $K = 50000$, $M = 32$. The actor network we use has two hidden layers with sizes 20 and 15, and the learning rate $\eta_\theta = 0.00001$. We build the critic network using the same structure with learning rate is $\eta_\beta = 0.0001$. We also bound the bidding at each time step as 1.5 \$/0.05 s to scale the output of DDPG. Our implementation is based on an open-source package DDPG.[3]

5.3.2. Training results

We train the actor and critic networks with the simulation setup as described above. The training history of the bidding policy for the four applications is shown in Fig. 7. The figure shows the total rewards (line is the average value and shade is the standard deviation) over 20 testing episodes from every 100 training episodes. As we can observe, the best bidding policy for application 2 is not bidding since the cost of bidding so that the deadline is not missing is more than the deadline missing penalty.

However, the bidding policies for tasks 1, 3, and 4 do not converge. The reason is that as we show in Section 4.2, there are two local minima in the cost function: no bidding, and minimum bidding for completing the job before deadline. Note that the second minimizer is unstable since a further small reduction on bidding may result in the convergence to the fist minimizer. So instead of using the DDPG model after the entire training, the final choice of our model is the best model during the training procedure based on the testing results as in Fig. 7.

Next, in order to validate the optimality of our obtained policy, we investigate the trained policies with fixed $P^- = 33$ \$/s and $\tau = 0.1$ s so that we can compare with the analytical form of best bidding in Section 4.2. The results and the comparison are listed in Table 3. We can see that DDPG almost captures either of the two local minima, depending on the amount of penalty. We also estimate the equivalent bidding rate as

$$\hat{p} = \frac{\sum_{t=1}^{l} a_t}{d - \tau}, \tag{29}$$

so we can compare it with the optimal bidding rate given in (20). We can see there is a small difference between $\hat{p}$ and p^*, which may be due to the fact that we discretize the time horizon and round up the completion time to steps of 0.05 s. Fig. 8 illustrates the detailed bidding policy at each time step for each application, where the y axis shows the actual bid per step instead of the bidding rate for easier comparison. We can see that DDPG tends to complete the process with fewer time steps but to split the bids equally among these steps.

5.3.3. Sensitivity analysis

In this subsection, we perform sensitivity analysis of different parameters, i.e., investigate how the bidding policy changes over task parameter variations. We choose application 2 to analyze how workload, deadline, and deadline-missing penalty can influence the obtained bidding policy. We first fix $w = 0.06$ million instruction, $d = 0.4$ s, and let $M = 2, 2.5, 3, 3.5$ \$, respectively. The results are shown in Fig. 9. We can see as the penalty increases, the bidding strategy switches from zero-bidding to minimum bidding for job completion. However, when $M = 2.5$\$, the total rewards of these two bidding strategies are very close, so the learned policy is slightly worse than the optimal policy with total bid 2.06\$ and assigned workload 0.06 million instructions. When M increases further, the learned policy becomes stable and optimal.

Next, we fix $d = 0.4$ s, $M = 3.5$ \$, and change $w = 0.06, 0.08, 0.1, 0.12$ million instruction, respectively. The results are shown in Fig. 10. We can see as the workload increases, the total bid also increases accordingly. When the total bid and QoS cost becomes

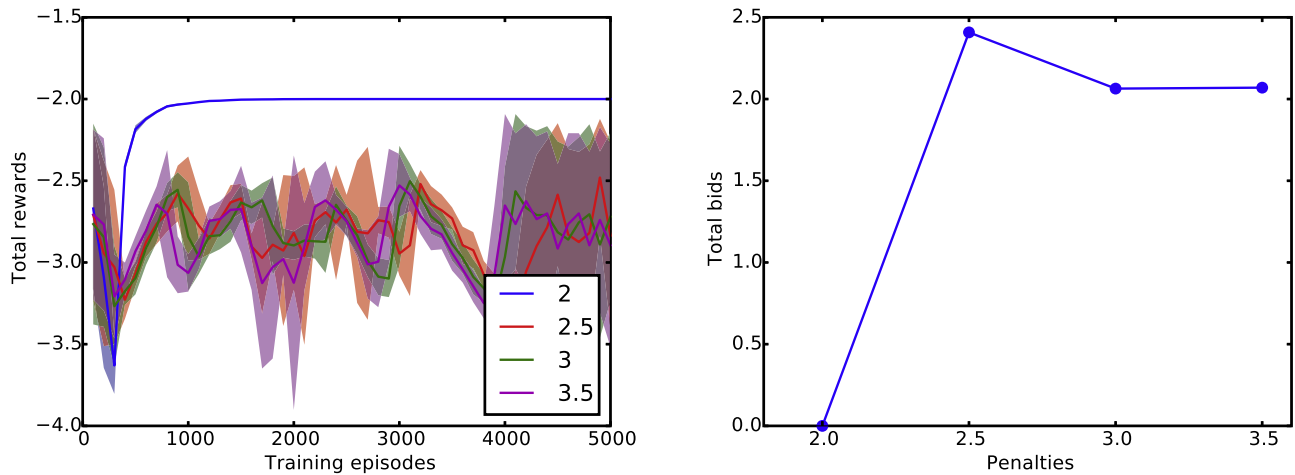Fig. 9. The impact of penalty on trained policy (in $). Left: rewards during training procedure, right: total bid vs. penalty.
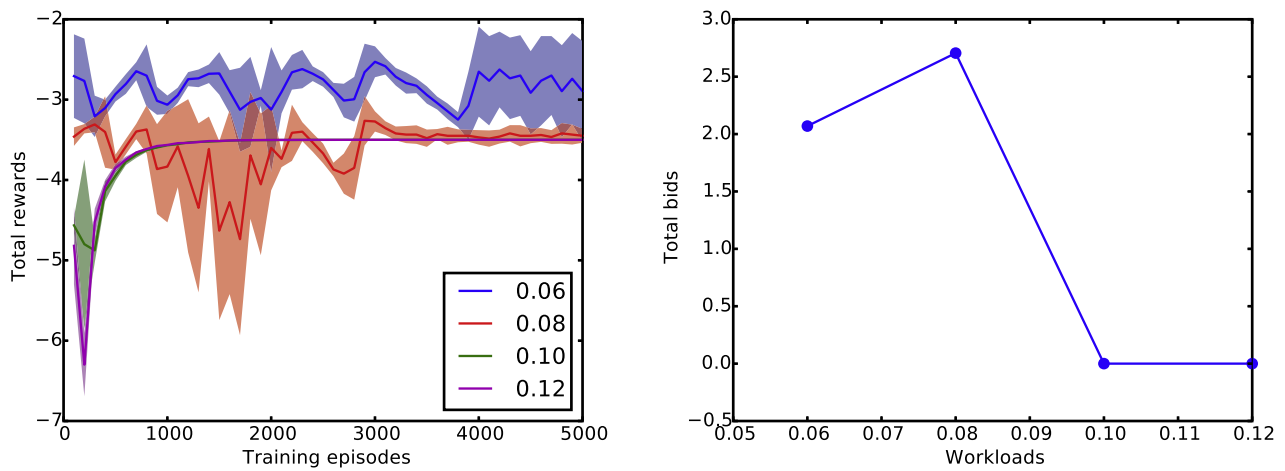

Fig. 10. The impact of workload on trained policy (in million instructions). Left: rewards during training procedure, right: total bid vs. workload.
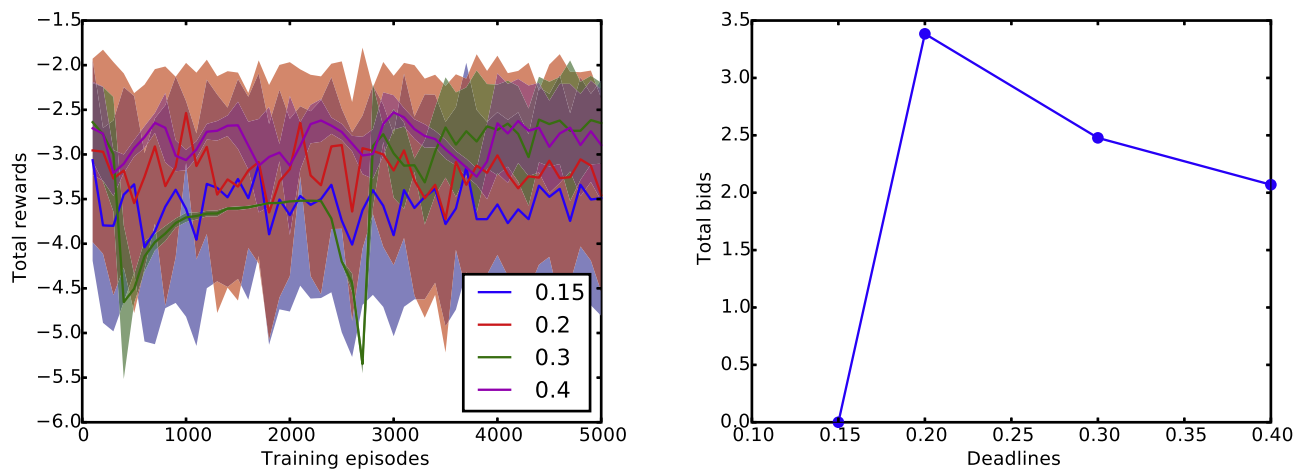

Fig. 11. The impact of deadline on trained policy (in seconds). Left: rewards during training procedure, right: total bid vs. deadline.

larger than the penalty, the bidding strategy switches to zero-bidding.

Finally, we fix $w = 0.06$ million instruction, $M = 3.5$ \$, and change $d = 0.15, 0.2, 0.3, 0.4$ s. Similar results can be observed in Fig. 11: when the deadline is long enough, DDPG always bids, when the deadline is so short ($d - \tau$ may be a single step) that the application can not be completed even with the maximum bid bound, DDPG switches to zero-bidding.

6. Conclusions

In this paper, we studied the problem of resource allocation for cloud-based automotive systems. Resource provisioning under both private and public cloud paradigms were modeled and treated. Task deadlines and random communication delays were explicitly considered in these models. In particular, a centralized resource provisioning scheme was used to model the dynamics of private cloud

provisioning and chance-constrained optimization was exploited to utilize the cloud resource to minimize the Quality of Service (QoS) cost while satisfying specified chance constraints. A decentralized auction-based bidding scheme was developed to model the public cloud resource provisioning. Best dynamics with constant bidding and constant time delays were first derived and a deep deterministic policy gradient was exploited to obtain the best bidding policy with random time delays and no prior knowledge on the random bidding from other vehicles. Numerical examples were presented to demonstrate the developed framework. We showed how the optimal bidding policy changes with parameters such as task workload and deadline.

Future work will include a demonstration with real-world vehicle applications. We will also consider a game theoretical formulation of the autonotive resource allocation problem in our future work.

References

[1] Amazon EC2. https://aws.amazon.com/ec2/pricing/on-demand/; Accessed: 2016-12-24.
[2] Ardagna D, Panicucci B, Passacantando M. A game theoretic formulation of the service provisioning problem in cloud systems. Proceedings of the 20th international conference on World wide web. ACM; 2011. p. 177–86.
[3] Ardagna D, Panicucci B, Passacantando M. Generalized nash equilibria for the service provisioning problem in cloud systems. IEEE Trans Serv Comput 2013;6(4):429–42. http://dx.doi.org/10.1109/TSC.2012.14.
[4] Bajcinca N. Wireless cars: a cyber-physical approach to vehicle dynamics control. Mechatronics 2015;30:261–74.
[5] Barshan M, Moens H, Turck FD. Design and evaluation of a scalable hierarchical application component placement algorithm for cloud resource allocation. 10th International conference on network and service management (CNSM) and workshop. 2014. p. 175–80. http://dx.doi.org/10.1109/CNSM.2014.7014155.
[6] Bellman R. A markovian decision process. Tech. Rep. DTIC Document; 1957.
[7] Dolgov M, Kurz G, Hanebeck UD. Chance-constrained model predictive control based on box approximations. 2015 54th IEEE conference on decision and control (CDC). 2015. p. 7189–94. http://dx.doi.org/10.1109/CDC.2015.7403353.
[8] Filev D, Lu J, Hrovat D. Future mobility: integrating vehicle control with cloud computing. Mech Eng 2013;135(3):S18.
[9] Hu X, Moura SJ, Murgovski N, Egardt B, Cao D. Integrated optimization of battery sizing, charging, and power management in plug-in hybrid electric vehicles. IEEE Trans Control Syst Technol 2016;24(3):1036–43. http://dx.doi.org/10.1109/TCST.2015.2476799.
[10] Ioffe S., Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:150203167 2015;.
[11] Johari R, Mannor S, Tsitsiklis JN. Efficiency loss in a network resource allocation game: the case of elastic supply. IEEE Trans Automat Contr 2005;50(11):1712–24. http://dx.doi.org/10.1109/TAC.2005.858687.
[12] Konda V, Tsitsiklis J. Actor-critic algorithms. NIPS. 13. 1999. p. 1008–14.
[13] Li Y, Tang X, Cai W. Dynamic bin packing for on-demand cloud resource allocation. IEEE Trans Parallel Distrib Syst 2016;27(1):157–70. http://dx.doi.org/10.1109/TPDS.2015.2393868.
[14] Li Z. Developments in estimation and control for cloud-enabled automotive vehicles. The University of Michigan; 2016. Ph.D. thesis.
[15] Li Z, Kolmanovsky I, Atkins E, Lu J, Filev D, Michelini J. Cloud aided semi-active suspension control. Computational intelligence in vehicles and transportation systems (CIVTS), 2014 IEEE symposium on. 2014. p. 76–83. http://dx.doi.org/10.1109/CIVTS.2014.7009481.
[16] Li Z, Kolmanovsky IV, Atkins EM, Lu J, Filev DP, Bai Y. Road disturbance estimation and cloud-aided comfort-based route planning. IEEE Trans Cybern 2016;PP (99):1–13. http://dx.doi.org/10.1109/TCYB.2016.2587673.
[17] Liang H, Cai L, Huang D, Shen X, Peng D. An SMDP-based service model for interdomain resource allocation in mobile cloud networks. IEEE Trans Veh Technol 2012;61(5):2222–32.
[18] Lillicrap T., Hunt J., Pritzel A., Heess N., Erez T., Tassa Y., et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971 2015;.
[19] Menache I, Ozdaglar A, Shimkin N. Socially optimal pricing of cloud computing resources. Proceedings of the 5th international ICST conference on performance evaluation methodologies and tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering); 2011. p. 322–31.
[20] Mnih V, Kavukcuoglu K, Silver D, Rusu A, Veness J, Bellemare M, et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33.
[21] Ono M. Closed-loop chance-constrained mpc with probabilistic resolvability. 2012 IEEE 51st IEEE conference on decision and control (CDC). 2012. p. 2611–8. http://dx.doi.org/10.1109/CDC.2012.6427393.
[22] Ozatay E, Onori S, Wollaeger J, Ozguner U, Rizzoni G, Filev D, et al. Cloud-based velocity profile optimization for everyday driving: a dynamic-programming-based solution. IEEE Trans Intell Transp Syst 2014;15(6):2491–505. http://dx.doi.org/10.1109/TITS.2014.2319812.
[23] Schwarm AT, Nikolaou M. Chance-constrained model predictive control. AIChE J 1999;45(8):1743–52. http://dx.doi.org/10.1002/aic.690450811.
[24] Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M. Deterministic policy gradient algorithms. In: Jebara T, Xing EP, editors. Proceedings of the 31st international conference on machine learning (ICML-14). JMLR Workshop and Conference Proceedings; 2014. p. 387–95http://jmlr.org/proceedings/papers/v32/silver14.pdf.
[25] Sookhak M, Yu F, Tang H. Secure data sharing for vehicular ad-hoc networks using cloud computing. Ad Hoc Networks. Springer; 2017. p. 306–15.
[26] Sutton R, Barto A. Reinforcement learning: an introduction. Neural Netw IEEE Trans 1998;9(5):1054.
[27] Sutton R, McAllester D, Singh S, Mansour Y, et al. Policy gradient methods for reinforcement learning with function approximation. NIPS. 99. 1999. p. 1057–63.
[28] Szepesvári C. Algorithms for reinforcement learning. Artif Intell Mach Learn 2010;4(1):1–103. Synthesis Lectures on
[29] Uhlenbeck G, Ornstein L. On the theory of the brownian motion. Phys Rev 1930;36(5):823.
[30] Whaiduzzaman M, Sookhak M, Gani A, Buyya R. A survey on vehicular cloud computing. J Netw Comput Appl 2014;40:325–44.
[31] Yan G, Wen D, Olariu S, Weigle M. Security challenges in vehicular cloud computing. IEEE Trans Intell Transp Syst 2013;14(1):284–94.
[32] Zheng K, Meng H, Chatzimisios P, Lei L, Shen X. An SMDP-based resource allocation in vehicular cloud computing systems. IEEE Trans Ind Electron 2015;62(12):7920–8.
[33] Zhou Z, Bambos N. A general model for resource allocation in utility computing. 2015 American control conference (ACC). 2015. p. 1746–51. http://dx.doi.org/10.1109/ACC.2015.7170985.