Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs

Shiqiang Wang, Rahul Urgaonkar, Ting He, *Senior Member, IEEE*, Kevin Chan, Murtaza Zafer, and Kin K. Leung, *Fellow, IEEE*

Abstract—Mobile micro-clouds are promising for enabling performance-critical cloud applications. However, one challenge therein is the dynamics at the network edge. In this paper, we study how to place service instances to cope with these dynamics, where multiple users and service instances coexist in the system. Our goal is to find the optimal placement (configuration) of instances to minimize the average cost over time, leveraging the ability of predicting future cost parameters with known accuracy. We first propose an offline algorithm that solves for the optimal configuration in a specific look-ahead time-window. Then, we propose an online approximation algorithm with polynomial time-complexity to find the placement in real-time whenever an instance arrives. We analytically show that the online algorithm is $O(1)$-competitive for a broad family of cost functions. Afterwards, the impact of prediction errors is considered and a method for finding the optimal look-ahead window size is proposed, which minimizes an upper bound of the average actual cost. The effectiveness of the proposed approach is evaluated by simulations with both synthetic and real-world (San Francisco taxi) user-mobility traces. The theoretical methodology used in this paper can potentially be applied to a larger class of dynamic resource allocation problems.

Index Terms—Cloud computing, fog/edge computing, online approximation algorithm, optimization, resource allocation, wireless networks

1 Introduction

Many emerging applications, such as video streaming, real-time face/object recognition, require high data processing capability. However, portable devices (e.g., smartphones) are generally limited by their size and battery life, which makes them incapable of performing complex computational tasks. A remedy for this is to utilize cloud computing techniques, where the cloud performs the computation for its users. In the traditional setting, cloud services are provided by centralized data-centers that may be located *far away* from end-users, which can be inefficient because users may experience long latency and poor connectivity due to long-distance communication [2]. The newly emerging idea of mobile micro-clouds (MMCs) is to place the cloud *closer* to end-users, so that users can have fast and reliable access to services. A small-sized server cluster hosting an MMC is directly connected to a network component at the network edge. For example, it can be connected to the wireless basestation, as proposed in [3] and [4], providing cloud services to users that are either connected to the basestation or are within a reasonable distance from it. It can also be connected to other network entities that are in close proximity to users. Fig. 1 shows an application scenario where MMCs coexist with a backend cloud. MMCs can be used for many applications that require high reliability or high data processing capability [2]. The same concept has also been termed as cloudlet [2], follow me cloud [5], fog computing, edge computing [6], small cell cloud [7], etc. We use the term MMC in this paper.

One important issue in MMCs is to decide which MMC should perform the computation for a particular user or a set of users, taking into account user mobility and other dynamic changes in the network. Providing a service to a user (or a set of users) requires starting a service instance, which can be run either in the backend cloud or in one of the MMCs, and the question is how to choose the optimal location to run the service instance. Besides, users may move across different geographical areas due to mobility, thus another question is whether we should migrate the service instance from one cloud (which can be either an MMC or the backend cloud) to another cloud when the user location or network condition changes. For every cloud, there is a cost[1] associated with running the service instance in it, and there is also a cost associated with migrating the service instance from one cloud to another cloud. The placement and migration of service instances therefore needs to properly take into account this cost.

- *S. Wang is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, United States. E-mail: wangshiq@us.ibm.com.*
- *R. Urgaonkar is with Amazon Inc., Seattle, WA 98109. E-mail: rahul.urgaonkar@gmail.com.*
- *T. He is with the School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA 16802, United States. E-mail: tzh58@psu.edu.*
- *K. Chan is with the Army Research Laboratory, Adelphi, MD 20783. E-mail: kevin.s.chan.civ@mail.mil.*
- *M. Zafer is with Nyansa Inc., Palo Alto, CA 94301. E-mail: murtaza.zafer.us@ieee.org.*
- *K.K. Leung is with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, United Kingdom. E-mail: kin.leung@imperial.ac.uk.*

Manuscript received 17 Sept. 2015; revised 5 July 2016; accepted 23 Aug. 2016. Date of publication 31 Aug. 2016; date of current version 15 Mar. 2017. Recommended for acceptance by U.V. Catalyurek.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2604814

1. The term "cost" in this paper is an abstract notion that can stand for monetary cost, service access latency of users, service interruption time, amount of transmission/processing resource consumption, etc.

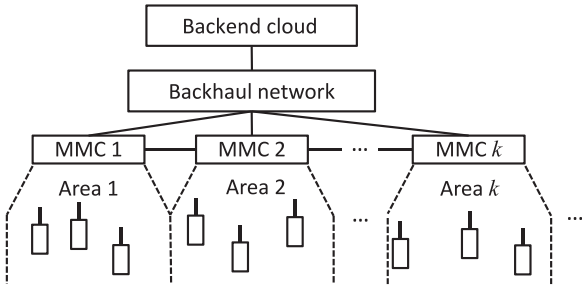


Fig. 1. Application scenario.

1.1 Related Work

The abovementioned problems are related to application/workload placement problems in cloud environments. Although existing work has studied such problems under complex network topologies [8], [9], they mainly focused on relatively static network conditions and fixed resource demands in a data-center environment. The presence of dynamically changing resource availability that is related to user mobility in an MMC environment has not been sufficiently considered.

When user mobility exists, it is necessary to consider real-time (live) migration of service instances. For example, it can be beneficial to migrate the instance to a location closer to the user. Only a few existing papers in the literature have studied this problem [10], [11], [12]. The main approach in [10], [11], [12] is to formulate the mobility-driven service instance migration problem as a Markov decision process (MDP). Such a formulation is suitable where the user mobility follows or can be approximated by a mobility model that can be described by a Markov chain. However, there are cases where the Markovian assumption is not valid [13]. Besides, [10], [11], [12] either do not explicitly or only heuristically consider multiple users and service instances, and they assume specific structures of the cost function that are related to the locations of users and service instances. Such cost structures may be inapplicable when the load on different MMCs are imbalanced or when we consider the backend cloud as a placement option. In addition, the existing MDP-based approaches mainly consider service instances that are constantly running in the cloud system; they do not consider instances that may arrive to and depart from the system over time.

Systems with online (and usually unpredictable) arrivals and departures have been studied in the field of online approximation algorithms [14], [15]. The goal is to design efficient algorithms (usually with polynomial time-complexity) that have reasonable competitive ratios.[2] However, most existing work focus on problems that can be formulated as integer *linear* programs. Problems that have *convex but non-linear* objective functions have attracted attention only very recently [16], [17], where the focus is on online covering problems in which new constraints arrive over time. Our problem is different from the existing work in the sense that the online arrivals in our problem are abstracted as *change*in constraints (or, with a slightly different but equivalent formulation, adding new *variables*) instead of adding new constraints, and we consider the average cost over multiple timeslots. Meanwhile, online departures are not considered in [16], [17].

Concurrently with the work presented in this paper, we have considered non-realtime applications in [18], where users submit job requests that can be processed after some time. Different from [18], we consider users continuously connected to services in this paper, which is often the case for delay-sensitive applications (such as live video streaming). The technical approach in this paper is fundamentally different from that in [18]. Besides, a Markovian mobility model is still assumed in [18].

Another related problem is the load balancing in distributed systems, where the goal is to even out the load distribution across machines. Migration cost, future cost parameter prediction and the impact of prediction error are not considered in load balancing problems [9], [19], [20], [31], [32]. We consider all these aspects in this paper, and in addition, we consider a generic cost definition that can be defined to favor load balancing as well as other aspects.

We also note that existing online algorithms with provable performance guarantees are often of theoretical nature [14], [15], [16], [17], which may not be straightforward to apply in practical systems because these algorithms can be conceptually complex thus difficult to understand. At the same time, most online algorithms applied in practice are of heuristic nature without theoretically provable optimality guarantees [8]; the performance of such algorithms are usually evaluated under a specific experimentation setting (see references of [8]), thus they may perform poorly under other settings that possibly occur in practice [21]. For example, in the machine scheduling problem considered in [22], a greedy algorithm (which is a common heuristic) that works well in some cases does not work well in other cases. We propose a simple and practically applicable online algorithm with theoretically provable performance guarantees in this paper, and also verify its performance with simulation using both synthetic arrivals and real-world user traces.

1.2 Main Contributions

In this paper, we consider a general setting which allows heterogeneity in cost values, network structure, and mobility models. We assume that the cost is related to a finite set of parameters, which can include the locations and preferences of users, load in the system, database locations, etc. We focus on the case where there is an underlying mechanism to predict the future values of these parameters, and also assume that the prediction mechanism provides the most likely future values and an upper bound on possible deviation of the actual value from the predicted value. Such an assumption is valid for many prediction methods that provide guarantees on prediction accuracy. Based on the predicted parameters, the (predicted) future costs of each configuration can be found, in which each configuration represents one particular placement sequence of service instances.

With the above assumption, we formulate a problem of finding the optimal configuration of service instances that minimizes the average cost over time. We define a look-ahead window to specify the amount of time that we look (predict) into the future. The main contributions of this paper are summarized as follows:

2. We define the *competitive ratio* as the maximum ratio of the cost from the online approximation algorithm to the true optimal cost from offline placement.

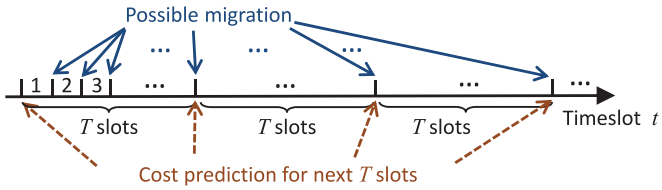


Fig. 2. Timing of the proposed approach.

1) We first focus on the *offline problem* of service instance placement using predicted costs within a specific look-ahead window, where the instance arrivals and departures within this look-ahead window are assumed to be known beforehand. We show that this problem is equivalent to a shortest-path problem in a virtual graph formed by all possible configurations, and propose an algorithm (Algorithm 2 in Section 3.3) to find its optimal solution using dynamic programming.
2) We note that it is often practically infeasible to know in advance about when an instance will arrive to or depart from the system. Meanwhile, Algorithm 2 may have exponential time-complexity when there exist multiple instances. Therefore, we propose an *online approximation algorithm* that finds the placement of a service instance upon its arrival with polynomial time-complexity. The proposed online algorithm calls Algorithm 2 as a subroutine for each instance upon its arrival. We analytically evaluate the performance of this online algorithm compared to the optimal offline placement. The proposed online algorithm is $O(1)$-competitive[3] for certain types of cost functions (including those which are linear, polynomial, or in some other specific form), under some mild assumptions.
3) Considering the existence of prediction errors, we propose a method to find the *optimal look-ahead window size*, such that an upper bound on the actual placement cost is minimized.
4) The effectiveness of the proposed approach is evaluated by *simulations* with both synthetic traces and *real-world mobility traces* of San Francisco taxis.

The remainder of this paper is organized as follows. The problem formulation is described in Section 2. Section 3 proposes an offline algorithm to find the optimal sequence of service instance placement with given look-ahead window size. The online placement algorithm and its performance analysis are presented in Section 4. Section 5 proposes a method to find the optimal look-ahead window size. Section 6 presents the simulation results and Section 7 draws conclusions.

2 Problem Formulation

We consider a cloud computing system as shown in Fig. 1, where the clouds are indexed by $k \in \{1, 2, \ldots, K\}$. Each cloud k can be either an MMC or a backend cloud. All MMCs together with the backend cloud can host service instances that may arrive and leave the system over time. A service instance is a process that is executed for a particular task of a cloud service. Each service instance *may serve one or a group users*, where there usually exists data transfer between the instance and the users it is serving. A time-slotted system as shown in Fig. 2 is considered, in which the actual physical time interval corresponding to each slot $t = 1, 2, 3, \ldots$ can be either the same or different.

We consider a window-based control framework, where every T slots, a controller performs cost prediction and computes the service instance configuration for the next T slots. We define these T consecutive slots as a *look-ahead window*. Service instance placement within each window is found either at the beginning of the window (in the offline case) or whenever an instance arrives (in the online case). We limit ourselves to within one look-ahead window when finding the configuration. In other words, we do not attempt to find the placement in the next window until the time for the current window has elapsed and the next window starts. Our solution can also be extended to a slot-based control framework where the controller computes the next T-slot configuration at the beginning of every slot, based on predicted cost parameters for the next T slots. We leave the detailed comparison of these frameworks and their variations for future work.

2.1 Definitions

We introduce some definitions in the following. A summary of main notations is given in Appendix A. *All appendices are included as part of the online supplementary material*, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2016.2604814.

2.1.1 Service Instances

We say a service instance *arrives* to the system if it is created, and we say it *departs* from the system if its operation is finished. Service instances may arrive and depart over time. We keep an index counter to assign an index for each new instance. The counter is initialized to zero when the cloud system starts to operate.[4] Upon a service instance arrival, we increment the counter by one, so that if the previously arrived instance has index i, a newly arrived instance will have index $i + 1$. With this definition, if $i < i'$, instance i arrives no later than instance i'. A particular instance i can only arrive to the system once, and we assume that arrivals always occur at the beginning of a slot and departures always occur at the end of a slot. For example, consider timeslots $t = 1, 2, 3, 4, 5$, instance $i = 2$ may arrive to the system at the beginning of slot $t = 2$, and depart from the system at the end of slot $t = 4$. At any timeslot t, instance i can have one of the following states: not arrived, running, or departed. For the above example, instance $i = 2$ has not yet arrived to the system in slot $t = 1$, it is running in slots $t = 2, 3, 4$, and it has already departed in slot $t = 5$. Note that an instance can be running across multiple windows each containing T slots before it departs.

2.1.2 Service Configurations

Consider an arbitrary sequence of consecutive timeslots $t \in \{t_0, t_0 + 1, \ldots, t_0 + Q - 1\}$, where Q is an integer. For simplicity, assume that the instance with the smallest index running in slot t_0 has index $i = 1$, and the instance with the largest index running in *any of* the slots in $\{t_0, \ldots, t_0 + Q - 1\}$

3. We say that an online algorithm is c-*competitive* if its competitive ratio is upper bounded by c.

4. This is for ease of presentation. In practice, the index can be reset when reaching a maximum counter number, and the definition of service configurations (defined later) can be easily modified accordingly.

has index M. According to the index assignment discussed in Section 2.1.1, there can be at most M instances running in any slot $t \in \{t_0, \ldots, t_0 + Q - 1\}$.

We define a Q-by-M matrix denoted by $\boldsymbol{\pi}$, where its (q,i)th ($q \in \{1, \ldots, Q\}$) element $(\boldsymbol{\pi})_{qi} \in \{0, 1, 2, \ldots, K\}$ denotes the location of service instance i in slot $t_q \triangleq t_0 + q - 1$ ("$\triangleq$" stands for "is defined to be equal to"). We set $(\boldsymbol{\pi})_{qi}$ according to the state of instance i in slot t_q, as follows

$$(\boldsymbol{\pi})_{qi} = \begin{cases} 0, & \text{if } i \text{ is not running in slot } t_q \\ k, & \text{if } i \text{ is running in cloud } k \text{ in slot } t_q, \end{cases}$$

where instance i is not running if it has not yet arrived or has already departed. The matrix $\boldsymbol{\pi}$ is called the *configuration* of instances in slots $\{t_0, \ldots, t_0 + Q - 1\}$. Throughout this paper, we use matrix $\boldsymbol{\pi}$ to represent configurations in different subsets of timeslots. We write $\boldsymbol{\pi}(t_0, t_1, \ldots, t_n)$ to explicitly denote the configuration in slots $\{t_0, t_1, \ldots, t_n\}$ (we have $Q = t_n - t_0 + 1$), and we write $\boldsymbol{\pi}$ for short where the considered slots can be inferred from the context. For a single slot t, $\boldsymbol{\pi}(t)$ becomes a vector (i.e., $Q = 1$).

Remark. The configurations in different slots can appear either in the same matrix or in different matrices. This means, from $\boldsymbol{\pi}(t_0, \ldots, t_0 + Q - 1)$, we can get $\boldsymbol{\pi}(t)$ for any $t \in \{t_0, \ldots, t_0 + Q - 1\}$, as well as $\boldsymbol{\pi}(t-1, t)$ for any $t \in \{t_0 + 1, \ldots, t_0 + Q - 1\}$, etc., and vice versa. For the ease of presentation later, we define $(\boldsymbol{\pi}(0))_i = 0$ for any i.

2.1.3 Costs

The *cost* can stand for different performance-related factors in practice, such as monetary cost (expressed as the price in some currency), service access latency of users (in seconds), service interruption time (in seconds), amount of transmission/processing resource consumption (in the number of bits to transfer, CPU cycles, memory size, etc.), or a combination of these. As long as these aspects can be expressed in some form of a cost function, we can treat them in the same optimization framework, thus we use the generic notion of cost in this paper.

We consider two types of costs. The *local cost* $U(t, \boldsymbol{\pi}(t))$ specifies the cost of data transmission (e.g., between each pair of user and service instance) and processing in slot t when the configuration in slot t is $\boldsymbol{\pi}(t)$. Its value can depend on many factors, including user location, network condition, load of clouds, etc., as discussed in Section 1.2. When a service instance is initiated in slot t, the local cost in slot t also includes the cost of initial placement of the corresponding service instance(s). We then define the *migration cost* $W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t))$, which specifies the cost related to migration between slots $t-1$ and t, which respectively have configurations $\boldsymbol{\pi}(t-1)$ and $\boldsymbol{\pi}(t)$. There is no migration cost in the very first timeslot (start of the system), thus we define $W(1, \cdot, \cdot) = 0$. The sum of local and migration costs in slot t when following configuration $\boldsymbol{\pi}(t-1, t)$ is given by

$$C_{\boldsymbol{\pi}(t-1,t)}(t) \triangleq U(t, \boldsymbol{\pi}(t)) + W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t)). \quad (1)$$

The above defined costs are *aggregated costs* for all service instances in the system during slot t. We will give concrete examples of these costs later in Section 4.2.1.

2.2 Actual and Predicted Costs

To distinguish between the actual and predicted cost values, for a given configuration $\boldsymbol{\pi}$, we let $A_{\boldsymbol{\pi}}(t)$ denote the *actual* value of $C_{\boldsymbol{\pi}}(t)$, and let $D_{\boldsymbol{\pi}}^{t_0}(t)$ denote the *predicted* (most likely) value of $C_{\boldsymbol{\pi}}(t)$, when cost-parameter prediction is performed at the beginning of slot t_0. For completeness of notations, we define $D_{\boldsymbol{\pi}}^{t_0}(t) = A_{\boldsymbol{\pi}}(t)$ for $t < t_0$, because at the beginning of t_0, the costs of all past timeslots are known. For $t \geq t_0$, we assume that the absolute difference between $A_{\boldsymbol{\pi}}(t)$ and $D_{\boldsymbol{\pi}}^{t_0}(t)$ is at most

$$\epsilon(\tau) \triangleq \max_{\boldsymbol{\pi}, t_0} \left| A_{\boldsymbol{\pi}}(t_0 + \tau) - D_{\boldsymbol{\pi}}^{t_0}(t_0 + \tau) \right|,$$

which represents the maximum error when looking ahead for τ slots, among all possible configurations $\boldsymbol{\pi}$ (note that only $\boldsymbol{\pi}(t_0 + \tau - 1)$ and $\boldsymbol{\pi}(t_0 + \tau)$ are relevant) and all possible prediction time instant t_0. The function $\epsilon(\tau)$ is assumed to be non-decreasing with τ, because we generally cannot have lower error when we look farther ahead into the future. The specific value of $\epsilon(\tau)$ is assumed to be provided by the cost prediction module.

We note that specific methods for predicting future cost parameters are beyond the scope of this paper, but we anticipate that existing approaches such as [23], [24], and [25] can be applied. For example, one simple approach is to measure cost parameters on the current network condition, and regard them as parameters for the future cost until the next measurement is taken. The prediction accuracy in this case is related to how fast the cost parameters vary, which can be estimated from historical records. We regard these cost parameters as predictable because they are generally related to the overall state of the system or historical pattern of users, which are unlikely to vary significantly from its previous state or pattern within a short time. This is *different* from arrivals and departures of instances, which can be spontaneous and unlikely to follow a predictable pattern.

2.3 Our Goal

Our ultimate goal is to find the optimal configuration $\boldsymbol{\pi}^*(1, \ldots, \infty)$ that minimizes the *actual* average cost over a sufficiently long time, i.e.,

$$\boldsymbol{\pi}^*(1, \ldots, \infty) = \arg \min_{\boldsymbol{\pi}(1,\ldots,\infty)} \lim_{T_{\max} \to \infty} \frac{\sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}(t-1,t)}(t)}{T_{\max}} \quad (2)$$

However, it is impractical to find the optimal solution to (2), because we cannot precisely predict the future costs and also do not have exact knowledge on instance arrival and departure events in the future. Therefore, we focus on obtaining an approximate solution to (2) by utilizing *predicted* cost values that are collected every T slots.

Now, the service placement problem includes two parts: one is finding the look-ahead window size T, discussed in Section 5; the other is finding the configuration within each window, where we consider both offline and online placements, discussed in Sections 3 (offline placement) and 4 (online placement). The offline placement assumes that at the beginning of window T, we know the exact arrival and departure times of each instance within the rest of window T, whereas the online placement does not assume this knowledge. We note that the notion of "offline" here does

not imply exact knowledge of future costs. Both offline and online placements in Sections 3 and 4 are based on the predicted costs $D_{\boldsymbol{\pi}}^{t_0}(t)$, the actual cost $A_{\boldsymbol{\pi}}(t)$ is considered later in Section 5.

3 Offline Service Placement with Given Look-Ahead Window Size

In this section, we focus on the offline placement problem, where the arrival and departure times of future instances are assumed to be exactly known. We denote the configuration found for this problem by $\boldsymbol{\pi}_{\text{off}}$.

3.1 Procedure

We start with illustrating the high-level procedure of finding $\boldsymbol{\pi}_{\text{off}}$. When the look-ahead window size T is given, the configuration $\boldsymbol{\pi}_{\text{off}}$ is found sequentially for each window (containing timeslots $t_0, \ldots, t_0+T-1$), by solving the following optimization problem:

$$\boldsymbol{\pi}_{\text{off}}(t_0, \ldots, t_0+T-1) = \arg \min_{\boldsymbol{\pi}(t_0,\ldots,t_0+T-1)} \sum_{t=t_0}^{t_0+T-1} D_{\boldsymbol{\pi}(t-1,t)}^{t_0}(t), \quad (3)$$

where $D_{\boldsymbol{\pi}}^{t_0}(t)$ can be found based on the parameters obtained from the cost prediction module. The procedure is shown in Algorithm 1.

In Algorithm 1, every time when solving (3), we get the value of $\boldsymbol{\pi}_{\text{off}}$ for additional T slots. This is sufficient in practice (compared to an alternative approach that directly solves for $\boldsymbol{\pi}_{\text{off}}$ for all slots) because we only need to know where to place the instances in the current slot. The value of $D_{\boldsymbol{\pi}(t-1,t)}^{t_0}(t)$ in (3) depends on the configuration in slot t_0-1, i.e., $\boldsymbol{\pi}(t_0-1)$, according to (1). When $t_0=1$, $\boldsymbol{\pi}(t_0-1)$ can be regarded as an arbitrary value, because the migration cost $W(t,\cdot,\cdot)=0$ for $t=1$.

Intuitively, at the beginning of slot t_0, (3) finds the optimal configuration that minimizes the predicted cost over the next T slots, given the locations of instances in slot t_0-1. We focus on solving (3) next.

Algorithm 1. Procedure of Offline Service Placement

1: Initialize $t_0 = 1$
2: **loop**
3: At the beginning of slot t_0, find the solution to (3)
4: Apply placements $\boldsymbol{\pi}_{\text{off}}(t_0, \ldots, t_0+T-1)$ in timeslots $t_0, \ldots, t_0+T-1$
5: $t_0 \leftarrow t_0+T$
6: **end loop**

3.2 Equivalence to Shortest-Path Problem

The problem in (3) is equivalent to a shortest-path problem with $D_{\boldsymbol{\pi}(t-1,t)}^{t_0}(t)$ as weights, as shown in Fig. 3. Each edge represents one possible combination of configurations in adjacent timeslots, and the weight on each edge is the predicted cost for such configurations. The configuration in slot t_0-1 is always given, and the number of possible configurations in subsequent timeslots is at most K^M, where M is defined as in Section 2.1.2 for the current window $\{t_0, \ldots, t_0+T-1\}$, and we note that depending on whether

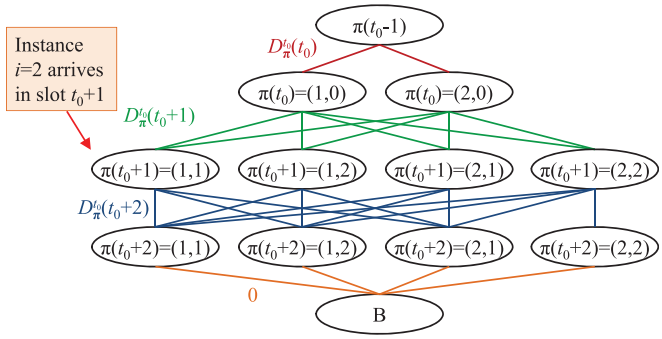


Fig. 3. Shortest-path formulation with $K=2$, $M=2$, and $T=3$. Instance $i=1$ is running in all slots, instance $i=2$ arrives at the beginning of slot t_0+1 and is running in slots t_0+1 and t_0+2.

the instance is running in the system or not, the number of possible configurations in a slot is either K or one (for configuration 0). Node B is a dummy node to ensure that we find a single shortest path, and the edges connecting node B have zero weights. It is obvious that the optimal solution to (3) can be found by taking the shortest (minimum-weighted) path from node $\boldsymbol{\pi}(t_0-1)$ to node B in Fig. 3; the nodes that the shortest path traverses correspond to the optimal solution $\boldsymbol{\pi}_{\text{off}}(t_0, \ldots, t_0+T-1)$ for (3).

3.3 Algorithm

We can solve the abovementioned shortest-path problem by means of dynamic programming [26]. The algorithm is shown in Algorithm 2, where we use $U_p(t, \mathbf{m})$ and $W_p(t, \mathbf{n}, \mathbf{m})$ to respectively denote the predicted local and migration costs when $\boldsymbol{\pi}(t)=\mathbf{m}$ and $\boldsymbol{\pi}(t-1)=\mathbf{n}$.

In the algorithm, Lines 5–16 iteratively find the shortest path (minimum objective function) for each timeslot. The iteration starts from the second level of the virtual graph in Fig. 3, which contains nodes with $\pi(t_0)$. It iterates through all the subsequent levels that respectively contain nodes with $\pi(t_0+1)$, $\pi(t_0+2)$, etc., excluding the last level with node B. In each iteration, the optimal solution for every possible (single-slot) configuration $\mathbf{m}$ is found by solving the Bellman's equation of the problem (Line 11). Essentially, the Bellman's equation finds the shortest path between the top node $\pi(t_0-1)$ and the current node under consideration (e.g., node $\pi(t_0+1)=(1,1)$ in Fig. 3), by considering all the nodes in the previous level (nodes $\pi(t_0)=(1,0)$ and $\pi(t_0)=(2,0)$ in Fig. 3). The sum weight on the shortest path between each node in the previous level and the top node $\pi(t_0-1)$ is stored in $\nu_{\mathbf{m}}$, and the corresponding nodes that this shortest path traverses through is stored in $\boldsymbol{\xi}_{\mathbf{m}}$. Based on $\nu_{\mathbf{m}}$ and $\boldsymbol{\xi}_{\mathbf{m}}$, Line 11 finds the shortest paths for all nodes in the current level, which can again be used for finding the shortest paths for all nodes in the next level (in the next iteration).

After iterating through all levels, the algorithm has found the shortest paths between top node $\pi(t_0-1)$ and all nodes in the last level with $\pi(t_0+T-1)$. Now, Lines 17 and 18 find the minimum of all these shortest paths, giving the optimal configuration. It is obvious that output of this algorithm satisfies the Bellman's principle of optimality, so the result is the shortest path and hence the optimal solution to (3).

Complexity. When the vectors $\boldsymbol{\pi}_{\mathbf{m}}$ and $\boldsymbol{\xi}_{\mathbf{m}}$ are stored as linked-lists, Algorithm 2 has time-complexity $O(K^{2M}T)$,

because the minimization in Line 11 requires enumerating at most K^M possible configurations, and there can be at most $K^M T$ possible combinations of values of t and $\mathbf{m}$.

Algorithm 2. Algorithm for Solving (3)

1: Define variables $\mathbf{m}$ and $\mathbf{n}$ to represent configurations respectively in the current and previous iteration (level of graph)
2: Define vectors $\boldsymbol{\pi}_{\mathbf{m}}$ and $\boldsymbol{\xi}_{\mathbf{m}}$ for all $\mathbf{m}, \mathbf{n}$, where $\boldsymbol{\pi}_{\mathbf{m}}$ (correspondingly, $\boldsymbol{\xi}_{\mathbf{m}}$) records the optimal configuration given that the configuration at the current (correspondingly, previous) timeslot of iteration is $\mathbf{m}$
3: Define variables $\mu_{\mathbf{m}}$ and $\nu_{\mathbf{m}}$ for all $\mathbf{m}$ to record the sum cost values from slot t_0 respectively to the current and previous slot of iteration, given that the configuration is $\mathbf{m}$ in the current or previous slot
4: Initialize $\mu_{\mathbf{m}} \leftarrow 0$ and $\boldsymbol{\pi}_{\mathbf{m}} \leftarrow \emptyset$ for all $\mathbf{m}$
5: **for** $t = t_0, \ldots, t_0 + T - 1$ **do**
6: **for** all $\mathbf{m}$ **do**
7: $\nu_{\mathbf{m}} \leftarrow \mu_{\mathbf{m}}$
8: $\boldsymbol{\xi}_{\mathbf{m}} \leftarrow \boldsymbol{\pi}_{\mathbf{m}}$
9: **end for**
10: **for** all $\mathbf{m}$ **do**
11: $\mathbf{n}^* \leftarrow \arg\min_{\mathbf{n}} \{\nu_{\mathbf{n}} + U_p(t, \mathbf{m}) + W_p(t, \mathbf{n}, \mathbf{m})\}$
12: $\boldsymbol{\pi}_{\mathbf{m}}(t_0, \ldots, t-1) \leftarrow \boldsymbol{\xi}_{\mathbf{n}^*}(t_0, \ldots, t-1)$
13: $\pi_{\mathbf{m}}(t) \leftarrow \mathbf{m}$
14: $\mu_{\mathbf{m}} \leftarrow \nu_{\mathbf{n}^*} + U_p(t, \mathbf{m}) + W_p(t, \mathbf{n}^*, \mathbf{m})$
15: **end for**
16: **end for**
17: $\mathbf{m}^* \leftarrow \arg\min_{\mathbf{m}} \mu_{\mathbf{m}}$
18: $\boldsymbol{\pi}_{\text{off}}(t_0, , t_0 + T - 1) \leftarrow \boldsymbol{\pi}_{\mathbf{m}^*}(t_0, \ldots, t_0 + T - 1)$
19: **return** $\boldsymbol{\pi}_{\text{off}}(t_0, \ldots, t_0 + T - 1)$

4 Complexity Reduction and Online Service Placement

The complexity of Algorithm 2 is exponential in the number of instances M, so it is desirable to reduce the complexity. In this section, we propose a method that can find an approximate solution to (3) and, at the same time, handle online instance arrivals and departures that are not known beforehand. We will also show that (3) is NP-hard when M is non-constant, which justifies the need to solve (3) approximately in an efficient manner.

4.1 Procedure

In the online case, we modify the procedure given in Algorithm 1 so that instances are placed one-by-one, where each placement greedily minimizes the objective function given in (3), while the configurations of previously placed instances remain unchanged.

We assume that each service instance i has a maximum lifetime $T_{\text{life}}(i)$, denoting the maximum number of remaining timeslots (including the current slot) that the instance remains in the system. The value of $T_{\text{life}}(i)$ may be infinity for instances that can potentially stay in the system for an arbitrary amount of time. The actual time that the instance stays in the system may be shorter than $T_{\text{life}}(i)$, but it cannot be longer than $T_{\text{life}}(i)$. When an instance leaves the system before its maximum lifetime has elapsed, we say that such a service instance departure is *unpredictable*.

We use $\boldsymbol{\pi}_{\text{on}}$ to denote the configuration $\boldsymbol{\pi}$ computed by online placement. The configuration $\boldsymbol{\pi}_{\text{on}}$ is updated every time when an instance arrives or unpredictably departs. At the beginning of the window (before any instance has arrived), it is initiated as an all-zero matrix.

For a specific look-ahead window $\{t_0, \ldots, t_0 + T - 1\}$, when service instance i arrives in slot $t \in \{t_0, \ldots, t_0 + T - 1\}$, we assume that this instance stays in the system until slot $t_e = \min\{t + T_{\text{life}}(i) - 1; t_0 + T - 1\}$, and accordingly update the configuration by

$$\boldsymbol{\pi}_{\text{on}}(t, \ldots, t_e) = \arg \min_{\boldsymbol{\pi}(t_a, \ldots, t_e)} \sum_{t=t_a}^{t_e} D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t) \quad \text{s.t.} \boldsymbol{\pi}(t, \ldots, t_e) = \boldsymbol{\pi}_{\text{on}}(t, \ldots, t_e) \text{ except for column } i. \tag{4}$$

Note that only the configuration of instance i (which is assumed to be stored in the ith column of $\boldsymbol{\pi}$) is found and updated in (4), the configurations of all other instances $i' \neq i$ remain unchanged. The solution to (4) can still be found with Algorithm 2. The only difference is that vectors $\mathbf{m}$ and $\mathbf{n}$ now become scalar values within $\{1, \ldots, K\}$, because we only consider the configuration of a single instance i. The complexity in this case becomes $O(K^2 T)$. At the beginning of the window, all the instances that have not departed after slot $t_0 - 1$ are seen as arrivals in slot t_0, because we independently consider the placements in each window of size T. When multiple instances arrive simultaneously, an arbitrary arrival sequence is assigned to them; the instances are still placed one-by-one by greedily minimizing (4).

Algorithm 3. Procedure of Online Service Placement

1: Initialize $t_0 = 1$
2: **loop**
3: Initialize $\boldsymbol{\pi}_{\text{on}}(t_0, \ldots, t_0 + T - 1)$ as an all-zero matrix
4: **for** each timeslot $t = t_0, \ldots, t_0 + T - 1$ **do**
5: **for** each instance i arriving at the beginning of slot t **do**
6: $t_e \leftarrow \min\{t + T_{\text{life}}(i) - 1; t_0 + T - 1\}$
7: Update $\boldsymbol{\pi}_{\text{on}}(t, \ldots, t_e)$ with the result from (4)
8: Apply configurations specified in the ith column of $\boldsymbol{\pi}_{\text{on}}(t, \ldots, t_e)$ for service instance i in timeslots $t, \ldots, t_e$ until instance i departs
9: **end for**
10: **for** each instance i departing at the end of slot t **do**
11: Set the ith column of $\boldsymbol{\pi}_{\text{on}}(t+1, \ldots, t_0 + T - 1)$ to zero
12: **end for**
13: **end for**
14: $t_0 \leftarrow t_0 + T$
15: **end loop**

When an instance i unpredictably departs at the end of slot $t \in \{t_0, \ldots, t_0 + T - 1\}$, we update $\boldsymbol{\pi}_{\text{on}}$ such that the ith column of $\boldsymbol{\pi}_{\text{on}}(t+1, \ldots, t_0 + T - 1)$ is set to zero.

The online procedure described above is shown in Algorithm 3. Recall that $\boldsymbol{\pi}_{\text{on}}(t, \ldots, t_e)$ and $\boldsymbol{\pi}_{\text{on}}(t+1, \ldots, t_0 + T - 1)$ are both part of a larger configuration matrix $\boldsymbol{\pi}_{\text{on}}(t_0, \ldots, t_0 + T - 1)$ (see Section 2.1.2).

Complexity. When placing a total of M instances, for a specific look-ahead window with size T, we can find the configurations of these M instances with complexity

$O(K^2TM)$, because (4) is solved for M times, each with complexity $O(K^2T)$.

Remark. It is important to note that in the above procedure, the configuration $\boldsymbol{\pi}_{\text{on}}$ (and thus the cost value $D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t)$ for *any* $t \in \{t_0, \ldots, t_0+T-1\}$) may vary upon instance arrival or departure. It follows that the T-slot sum cost $\sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}_{\text{on}}(t-1,t)}(t)$ may vary whenever an instance arrives or departs at an arbitrary slot $t \in \{t_0, \ldots, t_0+T-1\}$, and the value of $\sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}_{\text{on}}(t-1,t)}(t)$ stands for the predicted sum cost (over the current window containing T slots) under the *current* configuration, assuming that no new instance arrives and no instance unpredictably departs in the future. This variation in configuration and cost upon instance arrival/departure is frequently mentioned in the analysis presented next.

4.2 Performance Analysis

It is clear that for a single look-ahead window, Algorithm 3 has polynomial time-complexity while Algorithm 1 has exponential time-complexity. In this section, we show the NP-hardness of the offline service placement problem, and discuss the optimality gap between the online algorithm and the optimal offline placement. Note that we only focus on a single look-ahead window in this section. The interplay of multiple look-ahead windows and the impact of the window size will be considered in Section 5.

4.2.1 Definitions

For simplicity, we analyze the performance for a slightly restricted (but still general) class of cost functions. We introduce some additional definitions next (see Appendix A, available in the online supplemental material, for a summary of notations).

Indexing of Instances. Here, we assume that the instance with lowest index in the current window $\{t_0, \ldots, t_0+T-1\}$ has index $i=1$, and the last instance that arrives before the *current time of interest* has index $i=M$, where the current time of interest can be any time within the current window. With this definition, M does *not* need to be the largest index in window $\{t_0, \ldots, t_0+T-1\}$. Instead, it can be the index of *any* instance that arrives within $\{t_0, \ldots, t_0+T-1\}$. The cost of placing up to (and including) instance M is considered, where some instances $i \leq M$ may have already departed from the system.

Possible Configuration Sequence. When considering a window of T slots, we define the set of all possible configurations of a single instance as a set of T-dimensional vectors $\Lambda \triangleq \{(\lambda_1, \ldots, \lambda_T) : \lambda_n \in \{0, 1, \ldots, K\}, \forall n \in \{1, \ldots, T\}$, where λ_n is non-zero for at most one block of consecutive values of $n\}$. We also define a vector $\boldsymbol{\lambda} \in \Lambda$ to represent one *possible configuration sequence* of a single service instance across these T consecutive slots. For any instance i, the ith column of configuration matrix $\boldsymbol{\pi}(t_0, \ldots, t_0+T-1)$ is equal to one particular value of $\boldsymbol{\lambda}$.

We also define a binary variable $x_{i\boldsymbol{\lambda}}$, where $x_{i\boldsymbol{\lambda}}=1$ if instance i is placed according to configuration sequence $\boldsymbol{\lambda}$ across slots $\{t_0, \ldots, t_0+T-1\}$ (i.e., the ith column of $\boldsymbol{\pi}(t_0, \ldots, t_0+T-1)$ is equal to $\boldsymbol{\lambda}$), and $x_{i\boldsymbol{\lambda}}=0$ otherwise. We always have $\sum_{\boldsymbol{\lambda}\in\Lambda} x_{i\boldsymbol{\lambda}} = 1$ for all $i \in \{1, \ldots, M\}$.

We note that the values of $x_{i\boldsymbol{\lambda}}$ may vary over time due to arrivals and unpredictable departures of instances, which can be seen from Algorithm 3 and by noting the relationship between $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$. Before instance i arrives, $x_{i\boldsymbol{\lambda}_0}=1$ for $\boldsymbol{\lambda}_0=[0, \ldots, 0]$ which contains all zeros, and $x_{i\boldsymbol{\lambda}}=0$ for $\boldsymbol{\lambda} \neq \boldsymbol{\lambda}_0$. Upon arrival of instance i, we have $x_{i\boldsymbol{\lambda}_0}=0$ and $x_{i\boldsymbol{\lambda}_1}=1$ for a particular $\boldsymbol{\lambda}_1$. When instance i unpredictably departs at slot t', its configuration sequence switches from $\boldsymbol{\lambda}_1$ to an alternative (but partly correlated) sequence $\boldsymbol{\lambda}'_1$ (i.e., $(\boldsymbol{\lambda}'_1)_t = (\boldsymbol{\lambda}_1)_t$ for $t \leq t'$ and $(\boldsymbol{\lambda}'_1)_t = 0$ for $t > t'$, where $(\boldsymbol{\lambda})_t$ denotes the tth element of $\boldsymbol{\lambda}$), according to Line 11 in Algorithm 3, after which $x_{i\boldsymbol{\lambda}_1}=0$ and $x_{i\boldsymbol{\lambda}'_1}=1$.

Resource Consumption. We assume that the costs are related to the resource consumption, and for the ease of presentation, we consider two types of resource consumptions. The first type is associated with serving user requests, i.e., data transmission and processing when a cloud is running a service instance, which we refer to as the *local resource consumption*. The second type is associated with migration, i.e., migrating an instance from one cloud to another cloud, which we refer to as the *migration resource consumption*.

If we know that instance i operates under configuration sequence $\boldsymbol{\lambda}$, then we know whether instance i is placed on cloud k in slot t, for any $k \in \{1, \ldots, K\}$ and $t \in \{t_0, \ldots, t_0+T-1\}$. We also know whether instance i is migrated from cloud k to cloud l ($l \in \{1, 2, \ldots, K\}$) between slots $t-1$ and t. We use $a_{i\boldsymbol{\lambda}k}(t) \geq 0$ to denote the local resource consumption at cloud k in slot t when instance i is operating under $\boldsymbol{\lambda}$, where $a_{i\boldsymbol{\lambda}k}(t)=0$ if $(\boldsymbol{\lambda})_t \neq k$. We use $b_{i\boldsymbol{\lambda}kl}(t) \geq 0$ to denote themigration resource consumption when instance i operating under $\boldsymbol{\lambda}$ is assigned to cloud k in slot $t-1$ and to cloud l in slot t, where $b_{i\boldsymbol{\lambda}kl}(t)=0$ if $(\boldsymbol{\lambda})_{t-1} \neq k$ or $(\boldsymbol{\lambda})_t \neq l$, and we note that the configuration in slot t_0-1 (before the start of the current window) is assumed to be given and thus independent of $\boldsymbol{\lambda}$. The values of $a_{i\boldsymbol{\lambda}k}(t)$ and $b_{i\boldsymbol{\lambda}kl}(t)$ are either service-specific parameters that are known beforehand, or they can be found as part of the cost prediction.

We denote the sum local resource consumption at cloud k by $y_k(t) \triangleq \sum_{i=1}^{M} \sum_{\boldsymbol{\lambda}\in\Lambda} a_{i\boldsymbol{\lambda}k}(t) x_{i\boldsymbol{\lambda}}$, and denote the sum migration resource consumption from cloud k to cloud l by $z_{kl}(t) \triangleq \sum_{i=1}^{M} \sum_{\boldsymbol{\lambda}\in\Lambda} b_{i\boldsymbol{\lambda}kl}(t) x_{i\boldsymbol{\lambda}}$. We may omit the argument t in the following discussion.

Remark. The local and migration resource consumptions defined above can be related to CPU and communication bandwidth occupation, etc., or the sum of them. We only consider these two types of resource consumption for the ease of presentation. By applying the same theoretical framework, the performance gap results (presented later) can be extended to incorporate multiple types of resources and more sophisticated cost functions, and similar results yield for the general case.

Costs. We refine the costs defined in Section 2.1.3 by considering the cost for each cloud or each pair of clouds. The local cost at cloud k in timeslot t is denoted by $u_{k,t}(y_k(t))$. When an instance is initiated in slot t, the local cost in slot t also includes the cost of initial placement of the corresponding instance. The migration costfrom cloud k to cloud l between slots $t-1$ and t is denoted by $w_{kl,t}(y_k(t-1), y_l(t), z_{kl}(t))$. Besides $z_{kl}(t)$, the migration cost is also related to $y_k(t-1)$ and $y_l(t)$, because additional processing may be

needed for migration, and the cost for such processing can be related to the current load at clouds k and l. The functions $u_{k,t}(y)$ and $w_{kl,t}(y_k, y_l, z_{kl})$ can be different for different slots t and different clouds k and l, and they can depend on many factors, such as network condition, background load of the cloud, etc. Noting that any constant term added to the cost function does not affect the optimal configuration, we set $u_{k,t}(0) = 0$ and $w_{kl,t}(0, 0, 0) = 0$. We also set $w_{kl,t}(\cdot, \cdot, 0) = 0$, because there is no migration cost if we do not migrate. There is also no migration cost at the start of the first timeslot, thus we set $w_{kl,t}(\cdot, \cdot, \cdot) = 0$ for $t = 1$. With these definitions, the aggregated costs $U(t, \boldsymbol{\pi}(t))$ and $W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t))$ can be explicitly expressed as

$$U(t, \boldsymbol{\pi}(t)) \triangleq \sum_{k=1}^{K} u_{k,t}(y_k(t)) \tag{5}$$

$$W(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t)) \triangleq \sum_{k=1}^{K}\sum_{l=1}^{K} w_{kl,t}\left(y_k(t-1), y_l(t), z_{kl}(t)\right), \tag{6}$$

We then assume that the following assumption is satisfied for the cost functions, which holds for a large class of practical cost functions, such as those related to the delay performance or load balancing [9].

Assumption 1: *Both $u_{k,t}(y)$ and $w_{kl,t}(y_k, y_l, z_{kl})$ are* convex non-decreasing *functions of y (or y_k, y_l, z_{kl}), satisfying:*

- $\frac{du_{k,t}}{dy}(0) > 0$
- $\frac{\partial w_{kl,t}}{\partial z_{kl}}(\cdot, \cdot, 0) > 0$ *for $t \geq 2$*

for all t, k, and l (unless stated otherwise), where $\frac{du_{k,t}}{dy}(0)$ denotes the derivative of $u_{k,t}$ with respect to (w.r.t.) y evaluated at $y = 0$, and $\frac{\partial w_{kl,t}}{\partial z_{kl}}(\cdot, \cdot, 0)$ denotes the partial derivative of $w_{kl,t}$ w.r.t. z_{kl} evaluated at $z_{kl} = 0$ and arbitrary y_k and y_l.

Vector Notation. To simplify the presentation, we use vectors to denote a collection of variables across multiple clouds, slots, or configuration sequences. For simplicity, we index each element in the vector with multiple indexes that are related to the index of the element, and use the general notion $(\mathbf{g})_{h_1 h_2}$ (or $(\mathbf{g})_{h_1 h_2 h_3}$) to denote the (h_1, h_2)th (or (h_1, h_2, h_3)th) element in an arbitrary vector $\mathbf{g}$. Because we know the range of each index, multiple indexes can be easily mapped to a single index. We regard each vector as a *single-indexed* vector for the purpose of vector concatenation (i.e., joining two vectors into one vector) and gradient computation later.

We define vectors $\mathbf{y}$ (with KT elements), $\mathbf{z}$ (with K^2T elements), $\mathbf{x}$ (with MK^T elements), $\mathbf{a}_{i\boldsymbol{\lambda}}$ (with KT elements), and $\mathbf{b}_{i\boldsymbol{\lambda}}$ (with K^2T elements), for every value of $i \in \{1, 2, \ldots, M\}$ and $\boldsymbol{\lambda} \in \Lambda$. Different values of i and $\boldsymbol{\lambda}$ correspond to different vectors $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$. The elements in these vectors are defined as follows:

$$(\mathbf{y})_{kt} \triangleq y_k(t), (\mathbf{z})_{klt} \triangleq z_{kl}(t), (\mathbf{x})_{i\boldsymbol{\lambda}} \triangleq x_{i\boldsymbol{\lambda}},$$

$$(\mathbf{a}_{i\boldsymbol{\lambda}})_{kt} \triangleq a_{i\boldsymbol{\lambda}k}(t), (\mathbf{b}_{i\boldsymbol{\lambda}})_{klt} \triangleq b_{i\boldsymbol{\lambda}kl}(t).$$

As discussed earlier in this section, $x_{i\boldsymbol{\lambda}}$ may unpredictably change over time due to arrivals and departures of service instances. It follows that the vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ may vary over time (recall that $\mathbf{y}$ and $\mathbf{z}$ are dependent on $\mathbf{x}$ by definition). The vectors $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$ are constant.

Alternative Cost Expression. Using the above definitions, we can write the sum cost of all T slots as follows

$$\widetilde{D}(\mathbf{x}) \triangleq \widetilde{D}(\mathbf{y}, \mathbf{z}) \triangleq \sum_{t=t_0}^{t_0+T-1} \Bigg[\sum_{k=1}^{K} u_{k,t}(y_k(t)) + \sum_{k=1}^{K}\sum_{l=1}^{K} w_{kl,t}(y_k(t-1), y_l(t), z_{kl}(t)) \Bigg], \tag{7}$$

where the cost function $\widetilde{D}(\cdot)$ can be expressed either in terms of $\mathbf{x}$ or in terms of $(\mathbf{y}, \mathbf{z})$. The cost function defined in (7) is equivalent to $\sum_{t=t_0}^{t_0+T-1} D^{t_0}_{\boldsymbol{\pi}(t-1,t)}(t)$, readers are also referred to the per-slot cost defined in (1) for comparison. The value of $\widetilde{D}(\mathbf{x})$ or, equivalently, $\widetilde{D}(\mathbf{y}, \mathbf{z})$ may vary over time due to service arrivals and unpredictable service instance departures as discussed above.

4.2.2 Equivalent Problem Formulation

With the above definitions, the offline service placement problem in (3) can be equivalently formulated as the following, where our goal is to find the optimal configuration for all service instances $1, 2, \ldots, M$ (we consider the offline case here where we know when each instance arrives and no instance will unpredictably leave after they have been placed):

$$\begin{aligned} \min_{\mathbf{x}} \quad & \widetilde{D}(\mathbf{x}) \\ s.t. \quad & \sum_{\boldsymbol{\lambda} \in \Lambda_i} x_{i\boldsymbol{\lambda}} = 1, \forall i \in \{1, 2, \ldots, M\} \\ & x_{i\boldsymbol{\lambda}} \in \{0, 1\}, \forall i \in \{1, 2, \ldots, M\}, \boldsymbol{\lambda} \in \Lambda_i, \end{aligned} \tag{8}$$

where $\Lambda_i \subseteq \Lambda$ is a subset of feasible configuration sequences for instance i, i.e., sequences that contain those vectors whose elements are non-zero starting from the slot at which i arrives and ending at the slot at which i departs, while all other elements of the vectors are zero.

We now show that (8), and thus (3), is NP-hard even in the offline case, which further justifies the need for an approximation algorithm for solving the problem.

Proposition 1 (NP-Hardness). *The problem in (8) in the offline sense, and thus (3), is NP-hard.*

Proof. Problem (8) can be reduced from the partition problem, which is known to be NP-complete [27, Corollary 15.28]. See Appendix B, available in the online supplemental material, for details. □

An online version of problem (8) can be constructed by updating Λ_i over time. When an arbitrary instance i has not yet arrived, we define Λ_i as the set containing an all-zero vector. After instance i arrives, we assume that it will run in the system until t_e (defined in Section 4.1), and update Λ_i to conform to the arrival and departure times of instance i (see above). After instance i departs, Λ_i can be further updated so that the configurations corresponding to all remaining slots are zero.

4.2.3 Performance Gap

As discussed earlier, Algorithm 3 solves (8) in a greedy manner, where each service instance i is placed to greedily

minimize in (8). In the following, we compare the result from Algorithm 3 with the true optimal result, where the optimal result assumes offline placement. We use $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ to denote the result from Algorithm 3, and use $\mathbf{x}^*$ and $(\mathbf{y}^*, \mathbf{z}^*)$ to denote the offline optimal result to (8).

Lemma 1 (Convexity of $\widetilde{D}(\cdot)$). *When Assumption 1 is satisfied, the cost function $\widetilde{D}(\mathbf{x})$ or, equivalently, $\widetilde{D}(\mathbf{y}, \mathbf{z})$ is a non-decreasing convex function w.r.t. $\mathbf{x}$, and it is also a non-decreasing convex function w.r.t. $\mathbf{y}$ and $\mathbf{z}$.*

Proof. According to Assumption 1, $u_{k,t}(y_k(t))$ and $w_{kl,t}(y_k(t-1), y_l(t), z_{kl}(t))$ are non-decreasing convex functions. Because $y_k(t)$ and $z_{kl}(t)$ are linear mappings of $x_{i\boldsymbol{\lambda}}$ with non-negative weights for any t, k, and l, and also because the sum of non-decreasing convex functions is still a non-decreasing convex function, the lemma holds [28, Section 3.2]. □

In the following, we use $\nabla_{\mathbf{x}}$ to denote the gradient w.r.t. *each element* in vector $\mathbf{x}$, i.e., the $(i, \boldsymbol{\lambda})$th element of $\nabla_{\mathbf{x}}\widetilde{D}(\mathbf{x})$ is $\frac{\partial \widetilde{D}(\mathbf{x})}{\partial x_{i\boldsymbol{\lambda}}}$. Similarly, we use $\nabla_{\mathbf{y},\mathbf{z}}$ to denote the gradient w.r.t. each element in vector $(\mathbf{y}, \mathbf{z})$, where $(\mathbf{y}, \mathbf{z})$ is a vector that concatenates vectors $\mathbf{y}$ and $\mathbf{z}$.

Proposition 2 (Performance Gap). *When Assumption 1 is satisfied, we have*

$$\widetilde{D}(\mathbf{x}) \leq \widetilde{D}(\phi\psi\mathbf{x}^*), \tag{9}$$

or, equivalently,

$$\widetilde{D}(\mathbf{y}, \mathbf{z}) \leq \widetilde{D}(\phi\psi\mathbf{y}^*, \phi\psi\mathbf{z}^*), \tag{10}$$

where ϕ and ψ are constants satisfying

$$\phi \geq \frac{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\big(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{z}_{\max} + \mathbf{b}_{i\boldsymbol{\lambda}}\big) \cdot (\mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{b}_{i\boldsymbol{\lambda}})}{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}(\mathbf{y}, \mathbf{z}) \cdot (\mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{b}_{i\boldsymbol{\lambda}})} \tag{11}$$

$$\psi \geq \frac{\nabla_{\mathbf{x}}\widetilde{D}(\mathbf{x}) \cdot \mathbf{x}}{\widetilde{D}(\mathbf{x})} = \frac{\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}(\mathbf{y}, \mathbf{z}) \cdot (\mathbf{y}, \mathbf{z})}{\widetilde{D}(\mathbf{y}, \mathbf{z})}. \tag{12}$$

for any i and $\boldsymbol{\lambda} \in \Lambda_i$, in which $\mathbf{y}_{\max}$ and $\mathbf{z}_{\max}$ respectively denote the maximum values of $\mathbf{y}$ and $\mathbf{z}$ (the maximum is taken element-wise) after any number of instance arrivals within slots $\{t_0, \ldots, t_0 + T - 1\}$ until the current time of interest (at which time the latest arrived instance has index M), $(\mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{b}_{i\boldsymbol{\lambda}})$ is a vector that concatenates $\mathbf{a}_{i\boldsymbol{\lambda}}$ and $\mathbf{b}_{i\boldsymbol{\lambda}}$, and "$\cdot$" denotes the dot-product.

Proof. See Appendix C, available in the online supplemental material. □

Remark. We note that according to the definition of M in Section 4.2.1, the bound given in Proposition 2 holds at any time of interest within slots $\{t_0, \ldots, t_0 + T - 1\}$, i.e., for any number of instances that has arrived to the system, where some of them may have already departed.

4.2.4 Intuitive Explanation to the Constants ϕ and ψ

The constants ϕ and ψ in Proposition 2 are related to "how convex" the cost function is. In other words, they are related to how fast the cost of placing a single instance changes under different amount of existing resource consumption. Fig. 4 shows an illustrative example, where we only consider one cloud and one timeslot (i.e., $t = 1$, $T = 1$, and $K = 1$). In this case, setting $\phi = \frac{d\widetilde{D}}{dy}(y_{\max} + a_{\max}) \big/ \frac{d\widetilde{D}}{dy}(y)$ satisfies (11), where $a_{\max}$ denotes the maximum resource consumption of a single instance. Similarly, setting $\psi = \frac{d\widetilde{D}}{dy}(y) \cdot y \big/ \widetilde{D}(y)$ satisfies (12). We can see that the values of ϕ and ψ need to be larger when the cost function is more convex. For the general case, there is a weighted sum in both the numerator and denominator in (11) and (12). However, when we look at a single cloud (for the local cost) or a single pair of clouds (for the migration cost) in a single timeslot, the above intuition still applies.

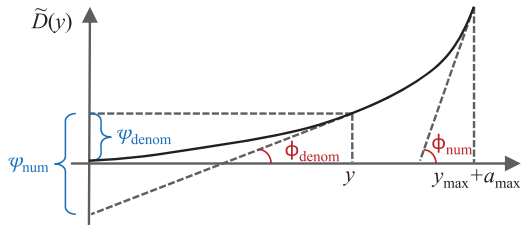


Fig. 4. Illustration of the performance gap for $t = 1$, $T = 1$, and $K = 1$, where $a_{\max}$ denotes the maximum resource consumption of a single instance. In this example, (11) becomes $\phi \geq \frac{\varphi_{\text{num}}}{\varphi_{\text{denom}}}$, and (12) becomes $\psi \geq \frac{\psi_{\text{num}}}{\psi_{\text{denom}}}$.

So, why is the optimality gap larger when the cost functions are more convex, i.e., have a larger second order derivative? We note that in the greedy assignment procedure in Algorithm 3, we choose the configuration of each instance i by minimizing the cost under the system state at the time when instance i arrives, where the system state represents the local and migration resource consumptions as specified by vectors $\mathbf{y}$ and $\mathbf{z}$. When cost functions are more convex, for an alternative system state $(\mathbf{y}', \mathbf{z}')$, it is more likely that the placement of instance i (which was determined at system state $(\mathbf{y}, \mathbf{z})$) becomes far from optimum. This is because if cost functions are more convex, the cost increase of placing a new instance i (assuming the same configuration for i) varies more when $(\mathbf{y}, \mathbf{z})$ changes. This intuition is confirmed by formal results described next.

4.2.5 Linear Cost Functions

Consider linear cost functions in the form of

$$u_{k,t}(y) = \gamma_{k,t} y \tag{13}$$

$$w_{kl,t}(y_k, y_l, z_{kl}) = \kappa^{(1)}_{kl,t} y_k + \kappa^{(2)}_{kl,t} y_l + \kappa^{(3)}_{kl,t} z_{kl}, \tag{14}$$

where the constants $\gamma_{k,t}, \kappa^{(3)}_{kl,t} > 0$ and $\kappa^{(1)}_{kl,t}, \kappa^{(2)}_{kl,t} \geq 0$.

Proposition 3. *When the cost functions are defined as in (13) and (14), Algorithm 3 provides the optimal solution.*

Proof. We have

$$\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}\big(\mathbf{y}_{\max} + \mathbf{a}_{i\boldsymbol{\lambda}}, \mathbf{z}_{\max} + \mathbf{b}_{i\boldsymbol{\lambda}}\big) = \nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}(\mathbf{y}, \mathbf{z})$$

$$\nabla_{\mathbf{y},\mathbf{z}}\widetilde{D}(\mathbf{y}, \mathbf{z}) \cdot (\mathbf{y}, \mathbf{z}) = \widetilde{D}(\mathbf{y}, \mathbf{z}),$$

because the gradient in this case is a constant. Hence, choosing $\phi = \psi = 1$ satisfies (11) and (12), yielding $\widetilde{D}(\mathbf{x}) \leq \widetilde{D}(\mathbf{x}^*)$ which means that the solution from Algorithm 3 is not worse than the optimal solution. □

This implies that the greedy service placement is optimal for linear cost functions, which is intuitive because the previous placements have no impact on the cost of later placements when the cost function is linear.

4.2.6 Polynomial Cost Functions

Consider polynomial cost functions in the form of

$$u_{k,t}(y) = \sum_{\rho} \gamma_{k,t}^{(\rho)} y^{\rho} \tag{15}$$

$$w_{kl,t}(y_k, y_l, z_{kl}) = \sum_{\rho_1} \sum_{\rho_2} \sum_{\rho_3} \kappa_{kl,t}^{(\rho_1,\rho_2,\rho_3)} y_k^{\rho_1} y_l^{\rho_2} z_{kl}^{\rho_3}, \tag{16}$$

where $\rho, \rho_1, \rho_2, \rho_3$ are integers satisfying $\rho \geq 1$, $\rho_1 + \rho_2 + \rho_3 \geq 1$ and the constants $\gamma_{k,t}^{(\rho)} \geq 0$, $\kappa_{kl,t}^{(\rho_1,\rho_2,\rho_3)} \geq 0$.

We first introduce the following assumption which can be satisfied in most practical systems with an upper bound on resource consumptions and departure rates.

Assumption 2. *The following is satisfied:*

- *For all $i, \boldsymbol{\lambda}, k, l, t$, there exists a constants $a_{\max}$ and $b_{\max}$, such that $a_{i\boldsymbol{\lambda}k}(t) \leq a_{\max}$ and $b_{i\boldsymbol{\lambda}kl}(t) \leq b_{\max}$*
- *The number of instances that unpredictably leave the system in each slot is upper bounded by a constant B_d.*

Proposition 4. *Assume that the cost functions are defined as in (15) and (16) while satisfying Assumption 1, and that Assumption 2 is satisfied.*

Let Ω denote the maximum value of ρ such that $\gamma_{k,t}^{(\rho)} > 0$ or $\kappa_{kl,t}^{(\rho_1,\rho_2,\rho_3)} > 0$, subject to $\rho_1 + \rho_2 + \rho_3 = \rho$. The value of Ω represents the highest order of the polynomial cost functions.

Define $\Gamma(\mathcal{I}(M)) \triangleq \widetilde{D}(\mathbf{x}_{\mathcal{I}(M)}) \Big/ \widetilde{D}(\mathbf{x}^*_{\mathcal{I}(M)})$, where $\mathcal{I}(M)$ is a problem input[5] containing M instances, and $\mathbf{x}_{\mathcal{I}(M)}$ and $\mathbf{x}^*_{\mathcal{I}(M)}$ are respectively the online and offline (optimal) results for input $\mathcal{I}(M)$. We say that Algorithm 3 is c-competitive in placing M instances if $\Gamma \triangleq \max_{\mathcal{I}(M)} \Gamma(\mathcal{I}(M)) \leq c$ for a given M. We have:

- *Algorithm 3 is $O(1)$-competitive.*
- *In particular, for any $\delta > 0$, there exists a sufficiently large M, such that Algorithm 3 is $\left(\Omega^{\Omega} + \delta\right)$-competitive.*

Proof. See Appendix D, available in the online supplemental material. □

Proposition 4 states that the competitive ratio does not indefinitely increase with increasing number of instances (specified by M). Instead, it approaches a constant value when M becomes large.

5. A particular problem input specifies the time each instance arrives/departs as well as the values of $\mathbf{a}_{i\lambda}$ and $\mathbf{b}_{i\lambda}$ for each i, λ.

When the cost functions are linear as in (13) and (14), we have $\Omega = 1$. In this case, Proposition 4 gives a competitive ratio upper bound of $1 + \delta$ (for sufficiently large M) where $\delta > 0$ can be arbitrarily small, while Proposition 3 shows that Algorithm 3 is optimal. This means that the competitive ratio upper bound given in Proposition 4 is *asymptotically tight* as M goes to infinity.

4.2.7 Linear Cost at Backend Cloud

Algorithm 3 is also $O(1)$-competitive for some more general forms of cost functions. For example, consider a simple case where there is no migration resource consumption, i.e., $b_{i\boldsymbol{\lambda}kl}(t) = 0$ for all $i, \boldsymbol{\lambda}, k, l$. Define $u_{k_0,t}(y) = \gamma y$ for some cloud k_0 and all t, where $\gamma > 0$ is a constant. For all other clouds $k \neq k_0$, define $u_{k,t}(y)$ as a general cost function while satisfying Assumption 1 and some additional mild assumptions presented below. Assume that there exists a constant $a_{\max}$ such that $a_{i\boldsymbol{\lambda}k}(t) \leq a_{\max}$ for all $i, \boldsymbol{\lambda}, k, t$.

Because $u_{k,t}(y)$ is convex non-decreasing and Algorithm 3 operates in a greedy manner, if $\frac{du_{k,t}}{dy}(y) > \gamma$, no new instance will be placed on cloud k, as it incurs higher cost than placing it on k_0. As a result, the maximum value of $y_k(t)$ is bounded, let us denote this upper bound by $y_k^{\max}(t)$. We note that $y_k^{\max}(t)$ is *only dependent on the cost function definition* and is independent of the number of arrived instances.

Assume $u_{k,t}(y_k^{\max}(t)) < \infty$ and $\frac{du_{k,t}}{dy}(y_k^{\max}(t) + a_{\max}) < \infty$ for all $k \neq k_0$ and t. When ignoring the cost at cloud k_0, the ratio $\Gamma(\mathcal{I}(M))$ does not indefinitely grow with incoming instances, because among all $y_k(t) \in [0, y_k^{\max}(t)]$ for all t and $k \neq k_0$, we can find ϕ and ψ that satisfy (11) and (12), we can also find the competitive ratio $\Gamma \triangleq \max_{\mathcal{I}(M)} \Gamma(\mathcal{I}(M))$. The resulting Γ is only dependent on the cost function definition, hence it does not keep increasing with M. Taking into account the cost at cloud k_0, the above result still applies, because the cost at k_0 is linear in $y_{k_0}(t)$, so that in either of (11), (12), or in the expression of $\Gamma(\mathcal{I}(M))$, the existence of this linear cost only adds a same quantity (which might be different in different expressions though) to both the numerator and denominator, which does not increase Γ (because $\Gamma \geq 1$).

The cloud k_0 can be considered as the backend cloud, which usually has abundant resources thus its cost-per-unit-resource often remains unchanged. This example can be generalized to cases with non-zero migration resource consumption, and we will illustrate such an application in the simulations in Section 6.

5 Optimal Look-Ahead Window Size

In this section, we study how to find the optimal window size T to look-ahead. When there are no errors in the cost prediction, setting T as large as possible can potentially bring the best long-term performance. However, the problem becomes more complicated when we consider the prediction error, because the farther ahead we look into the future, the less accurate the prediction becomes. When T is large, the predicted cost value may be far away from the actual cost, which can cause the configuration obtained from predicted costs $D_{\boldsymbol{\pi}}^{t_0}(t)$ with size-T windows (denoted by $\boldsymbol{\pi}_p$) deviate significantly from the true optimal

configuration $\boldsymbol{\pi}^*$ obtained from actual costs $A_{\boldsymbol{\pi}}(t)$. Note that $\boldsymbol{\pi}^*$ is obtained from actual costs $A_{\boldsymbol{\pi}}(t)$, which is different from $\mathbf{x}^*$ and $(\mathbf{y}^*, \mathbf{z}^*)$ which are obtained from predicted costs $D_{\boldsymbol{\pi}}^{t_0}(t)$ as defined in Section 4.2. Also note that $\boldsymbol{\pi}_p$ and $\boldsymbol{\pi}^*$ specify the configurations for an arbitrarily large number of timeslots, as in (2). Conversely, when T is small, the solution may not perform well in the long-term, because the look-ahead window is small and the long-term effect of migration is not considered. We have to find the optimal value of T which minimizes both the impact of prediction error and the impact of truncating the look-ahead time-span.

We assume that there exists a constant σ satisfying

$$\max_{\boldsymbol{\pi}(t-1,t)} W_a(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t)) \leq \sigma, \tag{17}$$

for any t, to represent the maximum value of the actual migration cost in any slot, where $W_a(t, \boldsymbol{\pi}(t-1), \boldsymbol{\pi}(t))$ denotes the actual migration cost. The value of σ is system-specific and is related to the cost definition.

To help with our analysis below, we define the sum-error starting from slot t_0 up to slot $t_0 + T - 1$ as

$$F(T) \triangleq \sum_{t=t_0}^{t_0+T-1} \epsilon(t - t_0). \tag{18}$$

Because $\epsilon(t - t_0) \geq 0$ and $\epsilon(t - t_0)$ is non-decreasing with t, it is obvious that $F(T+2) - F(T+1) \geq F(T+1) - F(T)$. Hence, $F(T)$ is a convex non-decreasing function for $T \geq 0$, where we define $F(0) = 0$.

5.1 Upper Bound on Cost Difference

In the following, we focus on the objective function given in (2), and study how worse the configuration $\boldsymbol{\pi}_p$ can perform, compared to the optimal configuration $\boldsymbol{\pi}^*$.

Proposition 5. *For look-ahead window size T, suppose that we can solve (3) with competitive ratio $\Gamma \geq 1$, the upper bound on the cost difference (while taking the competitive ratio Γ into account) from configurations $\boldsymbol{\pi}_p$ and $\boldsymbol{\pi}^*$ is given by*

$$\lim_{T_{\max}\to\infty} \left(\frac{\sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}_p}(t)}{T_{\max}} - \Gamma \frac{\sum_{t=1}^{T_{\max}} A_{\boldsymbol{\pi}^*}(t)}{T_{\max}} \right) \leq \frac{(\Gamma+1)F(T)+\sigma}{T}. \tag{19}$$

Proof. See Appendix E, available in the online supplemental material. □

We assume in the following that the competitive ratio Γ is independent of the choice of T, and regard it as a given parameter in the problem of finding optimal T. This assumption is justified for several cost functions where there exist a uniform bound on the competitive ratio for arbitrarily many services (see Sections 4.2.5, 4.2.6, and 4.2.7). We define the *optimal look-ahead window size* as the solution to the following optimization problem:

$$\begin{aligned} \min_{T} \quad & \frac{(\Gamma+1)F(T)+\sigma}{T} \\ \text{s.t.} \quad & T \geq 1. \end{aligned} \tag{20}$$

Considering the original objective in (2), the problem (20) can be regarded as finding the optimal look-ahead window size such that an upper bound of the objective function in (2) is minimized (according to Proposition 5). The solution to (20) is the optimal window size to look-ahead so that (in the worst case) the cost is closest to the cost of the optimal configuration $\boldsymbol{\pi}^*$.

5.2 Characteristics of the Problem in (20)

We now study the characteristics of (20). To help with the analysis, we interchangeably use variable T to represent either a discrete or a continuous variable. We define a continuous convex function $G(T)$, where $T \geq 1$ is a continuous variable. The function $G(T)$ is defined in such a way that $G(T) = F(T)$ for all the discrete values $T \in \{1, 2, \ldots\}$, i.e., $G(T)$ is a *continuous time extension* of $F(T)$. Such a definition is always possible by connecting the discrete points in $F(T)$. Note that we do not assume the continuity of the derivatives of $G(T)$, which means that $\frac{dG(T)}{dT}$ may be non-continuous and $\frac{d^2G(T)}{dT^2}$ may have $+\infty$ values. However, these do not affect our analysis below. We will work with continuous values of T in some parts and will discretize it when appropriate.

We define a function $\theta(T) \triangleq \frac{(\Gamma+1)G(T)+\sigma}{T}$ to represent the objective function in (20) after replacing $F(T)$ with $G(T)$, where T is regarded as a continuous variable. We take the logarithm of $\theta(T)$, yielding

$$\ln\theta = \ln((\Gamma+1)G(T) + \sigma) - \ln T. \tag{21}$$

Taking the derivative of $\ln\theta$, we have

$$\frac{d\ln\theta}{dT} = \frac{(\Gamma+1)\frac{dG(T)}{dT}}{(\Gamma+1)G(T)+\sigma} - \frac{1}{T}. \tag{22}$$

We set (22) equal to zero, and rearrange the equation, yielding

$$\Phi(T) \triangleq (\Gamma+1)T\frac{dG(T)}{dT} - (\Gamma+1)G(T) - \sigma = 0. \tag{23}$$

We have the following proposition and its corollary, their proofs are given in Appendix F, available in the online supplemental material.

Proposition 6. *Let T_0 denote a solution to (23), if the solution exists, then the optimal look-ahead window size T^* for problem (20) is either $\lfloor T_0 \rfloor$ or $\lceil T_0 \rceil$, where $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively denote the floor (rounding down to integer) and ceiling (rounding up to integer) of x.*

Corollary 1. *For window sizes T and $T+1$, if $\theta(T) < \theta(T+1)$, then the optimal size $T^* \leq T$; if $\theta(T) > \theta(T+1)$, then $T^* \geq T+1$; if $\theta(T) = \theta(T+1)$, then $T^* = T$.*

5.3 Finding the Optimal Solution

According to Proposition 6, we can solve (23) to find the optimal look-ahead window size. When $G(T)$ (and $F(T)$) can be expressed in some specific analytical forms, the solution to (23) can be found analytically. For example, consider $G(T) = F(T) = \beta T^\alpha$, where $\beta > 0$ and $\alpha > 1$. In this case, $T_0 = \left(\frac{\sigma}{(\Gamma+1)\beta(\alpha-1)}\right)^{\frac{1}{\alpha}}$, and $T^* = \arg\min_{T\in\{\lfloor T_0\rfloor, \lceil T_0\rceil\}} \theta(T)$. One can also use such specific forms as an upper bound for a general function.

When $G(T)$ (and $F(T)$) have more general forms, we can perform a search on the optimal window size according to the properties discussed in Section 5.2. Because we do not know the convexity of $\theta(T)$ or $\Phi(T)$, standard numerical methods for solving (20) or (23) may not be efficient. However, from Corollary 1, we know that the local minimum of $\theta(T)$ is the global minimum, so we can develop algorithms that use this property.

The optimal window size T^* takes discrete values, so we can perform a discrete search on $T \in \{1, 2, \ldots, T_m\}$, where $T_m > 1$ is a pre-specified upper limit on the search range. We then compare $\theta(T)$ with $\theta(T+1)$ and determine the optimal solution according to Corollary 1. One possible approach is to use binary search, as shown in Algorithm 4, which has time-complexity of $O(\log T_m)$.

Remark. The exact value of Γ may be difficult to find in practice, and (19) is an upper bound which may have a gap from the actual value of the left hand-side of (19). Therefore, in practice, we can regard Γ as a tuning parameter, which can be tuned so that the resulting window size T^* yields good performance. For a similar reason, the parameter σ can also be regarded as a tuning parameter in practice.

Algorithm 4. Binary Search for Finding Optimal Window Size

```
Initialize variables T_- ← 1 and T_+ ← T_m
repeat
T ← ⌊(T_- + T_+)/2⌋
if θ(T) < θ(T+1) then
T_+ ← T
else if θ(T) > θ(T+1) then
T_- ← T+1
else if θ(T) = θ(T+1) then
return T //Optimum found
end if
until T_- = T_+
return T_-
```

6 Simulation Results

In the simulations, we assume that there exist a backend cloud (with index k_0) and multiple MMCs. A service instance can be placed either on one of the MMCs or on the backend cloud. We first define

$$R(y) \triangleq \begin{cases} \frac{1}{1-\frac{y}{Y}}, & \text{if } y < Y \\ +\infty, & \text{if } y \geq Y, \end{cases} \tag{24}$$

where Y denotes the capacity of a single MMC. Then, we define the local and migration costs as in (5), (6), with

$$u_{k,t}(y_k(t)) \triangleq \begin{cases} \tilde{g} y_k(t), & \text{if } k = k_0 \\ y_k(t) R(y_k(t)) + g r_k(t), & \text{if } k \neq k_0 \end{cases} \tag{25}$$

$$w_{kl,t}(y_k(t-1), y_l(t), z_{kl}(t)) \triangleq \begin{cases} \tilde{h} z_{kl}(t), & \text{if } k = k_0 \text{ or/and } l = k_0 \\ z_{kl}(t)(R(y_k(t)) + R(y_l(t))) + h s_{kl}(t), & \text{else,} \end{cases} \tag{26}$$

where $y_k(t)$ and $z_{kl}(t)$ are sum resource consumptions defined as in Section 4.2.1, $r_k(t)$ is the sum of the distances between each instance running on cloud k and all users connected to this instance, $s_{kl}(t)$ is the distance between clouds k and l multiplied by the number migrated instances from cloud k to cloud l, and $\tilde{g}, g, \tilde{h}, h$ are simulation parameters (specified later). The *distance* here is expressed as the number of hops on the communication network.

Similar to Section 4.2.7 (but with migration cost here), we consider the scenario where the connection status to the backend cloud remains relatively unchanged. Thus, in (25) and (26), $u_{k,t}(\cdot)$ and $w_{kl,t}(\cdot,\cdot,\cdot)$ are linear in $y_k(t)$ and $z_{kl}(t)$ when involving the backend cloud k_0. When not involving the backend cloud, the cost functions have two terms. The first term contains $R(\cdot)$ and is related to the queuing delay of data processing/transmission, because $R(\cdot)$ has a similar form as the average queueing delay expression from queueing theory. The additional coefficient $y_k(t)$ or $z_{kl}(t)$ scales the delay by the total amount of workload so that experiences of all instances (hosted at a cloud or being migrated) are considered. This expression is also a widely used objective (such as in [9]) for pushing the system towards a load-balanced state. The second term has the distance of data transmission or migration, which is related to propagation delay. Thus, both queueing and propagation delays are captured in the cost definition above.

Note that the above defined cost functions are heterogeneous, because the cost definitions are different depending on whether the backend cloud is involved or not. Therefore, we cannot directly apply the existing MDP-based approaches [10], [11], [12] to solve this problem. We consider users continuously connected to service instances, so we also cannot apply the technique in [18].

6.1 Synthetic Arrivals and Departures

To evaluate how much worse the online placement (presented in Section 4) performs compared to the optimal offline placement (presented in Section 3), we first consider a setting with synthetic instance arrivals and departures. For simplicity, we ignore the migration cost and set $g = 0$ to make the local cost independent of the distance $r_k(t)$. We set $Y = 5$, $\tilde{g} = 3$, and the total number of clouds $K = 5$ among which one is the backend cloud. We simulate 4,000 arrivals, where the local resource consumption of each arrival is uniformly distributed within interval $[0.5, 1.5]$. Before a new instance arrives, we generate a random variable H that is uniformly distributed within $[0, 1]$. If $H < 0.1$, one randomly selected instance that is currently running in the system (if any) departs. We only focus on the cost in a single timeslot and assume that arrival and departure events happen within this slot. The online placement greedily places each instance, while the offline placement considers all instances as an entirety. We compare the cost of the proposed online placement algorithm with a lower bound of the cost of the optimal placement. The optimal lower bound is obtained by solving an optimization problem that allows every instance to be arbitrarily split across multiple clouds, in which case the problem becomes a convex optimization problem due to the relaxation of integer constraints.

The simulation is run with 100 different random seeds. Fig. 5 shows the overall results. We see that the cost is convex increasing when the number of arrived instances is

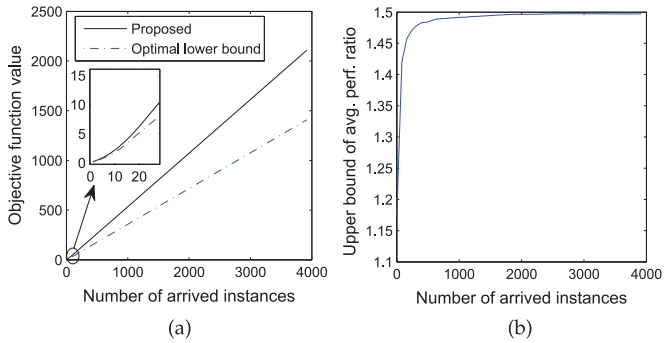


Fig. 5. Results with synthetic traces: (a) Objective function value, (b) average performance ratio.

small, and it increases linearly when the number of instances becomes large, because in the latter case, the MMCs are close to being overloaded and most instances are placed at the backend cloud. The fact that the average performance ratio (defined as $\mathrm{mean}(\widetilde{D}(\mathbf{x}))/\mathrm{mean}(\widetilde{D}(\mathbf{x}^*))$) converges with increasing number of instances in Fig. 5b supports our analysis in Section 4.2.7.

6.2 Real-World Traces

To further evaluate the performance while considering the impact of prediction errors and look-ahead window size, we perform simulations using real-world San Francisco taxi traces obtained on the day of May 31, 2008 [29], [30]. Similar to [12], we assume that the MMCs are deployed according to a hexagonal cellular structure in the central area of San Francisco (the center of this area has latitude 37.762 and longitude -122.43). The distance between the center points of adjacent cells is $1{,}000$ m. We consider $K-1=91$ cells (thus MMCs), one backend cloud, and 50 users (taxis) in total and not all the users are active at a given time. A user is considered active if its most recent location update was received within 600 s from the current time and its location is within the area covered by MMCs. Each user may require at most one service at a time from the cloud when it is active, where the duration that each active user requires (or, does not require) service is exponentially distributed with a mean value of 50 slots (or, 10 slots). When a user requires service, we assume that there is a service instance for this particular request (independent from other users) running on one of the clouds. The local and migration (if migration occurs) resource consumptions of each such instance are set to 1. We assume that the online algorithm has no knowledge on the departure time of instances and set $T_{\mathrm{life}}=\infty$ for all instances. Note that the taxi locations in the dataset are unevenly distributed, so it is still possible that one MMC hosts multiple services although the maximum possible number of instances (50) is smaller than the number of MMCs (91). The distance metric (for evaluating $r_k(t)$ and $s_{kl}(t)$) is defined as the minimum number of hops between two locations on the cellular structure. The physical time corresponding to each slot is set to 60 s. We set the parameters $\Gamma=1.5, \sigma=2, Y=5, \tilde{g}=\tilde{h}=3, g=h=0.2$. The cost prediction error is assumed to have an upper bound in the form of $F(T)=\beta T^{\alpha}$ (see Section 5.3), where we fix $\alpha=1.1$. The prediction error is generated randomly while ensuring that the upper bound is satisfied.

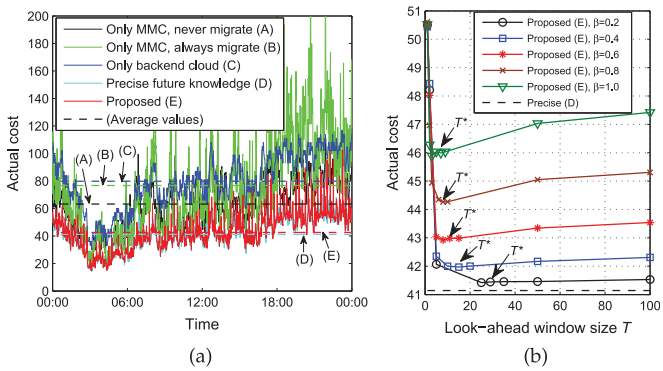


Fig. 6. Results with real-world traces (where the costs are summed over all clouds, i.e., the $A(t)$ values): (a) Actual costs at different time of a day, where $\beta=0.4$ for the proposed method E. The arrows point to the average values over the whole day of the corresponding policy. (b) Actual costs averaged over the whole day.

The simulation results are shown in Fig. 6. In Fig. 6a, we can see that the result of the proposed online placement approach (E) performs close to the case of online placement with precise future knowledge (D), where approach D assumes that all the future costs as well as instance arrival and departure times are precisely known, but we still use the online algorithm to determine the placement (i.e., we greedily place each instance), because the offline algorithm is too time consuming due to its high complexity. The proposed method E also outperforms alternative methods including only placing on MMCs and never migrate the service instance after initialization (A), always following the user when the user moves to a different cell (B), as well as always placing the service instance on the backend cloud (C). In approaches A and B, the instance placement is determined greedily so that the distance between the instance and its corresponding user is the shortest, subject to the MMC capacity constraint Y so that the costs are finite (see (24)). The fluctuation of the cost during the day is because of different number of users that require the service (thus different system load). In Fig. 6b, we show the average cost over the day with different look-ahead window sizes and β values (a large β indicates a large prediction error), where the average results from 8 different random seeds are shown. We see that the optimal window size (T^*) found from the method proposed in Section 5 is close to the window size that brings the lowest cost, which implies that the proposed method for finding T^* is reasonably accurate.

Additional results on the amount of computation time and floating-point operations (FLOP) for the results in Fig. 6a are given in Appendix G, available in the online supplemental material.

7 Conclusions

In this paper, we have studied the dynamic service placement problem for MMCs with multiple service instances, where the future costs are predictable within a known accuracy. We have proposed algorithms for both offline and online placements, as well as a method for finding the optimal look-ahead window size. The performance of the proposed algorithms has been evaluated both analytically and using simulations with synthetic instance arrival/departure

traces and real-world user mobility traces of San Francisco taxis. The simulation results support our analysis.

Our results are based on a general cost function that can represent different aspects in practice. As long as one can assign a cost for every possible configuration, the proposed algorithms are applicable, and the time-complexity results hold. The optimality gap for the online algorithm has been analyzed for a narrower class of functions, which is still very general as discussed earlier in the paper.

The theoretical framework used for analyzing the performance of online placement can be extended to incorporate more general cases, such as those where there exist multiple types of resources in each cloud. We envision that the performance results are similar. We also note that our framework can be applied to analyzing a large class of online resource allocation problems that have convex objective functions.

Acknowledgments

Contribution of S. Wang is related to his previous affiliation with Imperial College London. Contributions of R. Urgaonkar and T. He are related to their previous affiliation with IBM T.J. Watson Research Center. Contribution of M. Zafer is not related to his current employment at Nyansa Inc. This research was sponsored in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. A preliminary version of this paper was presented at IEEE ICC 2015 [1].

References

[1] S. Wang, R. Urgaonkar, K. Chan, T. He, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future cost," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2015, pp. 5504–5510.

[2] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: At the leading edge of mobile-cloud convergence," in *Proc. 6th Int. Conf. Mobile Comput. Appl. Serv.*, Nov. 2014, pp. 1–9.

[3] "Smarter wireless networks," *IBM Whitepaper No. WSW14201USEN*, Feb. 2013. [Online]. Available: www.ibm.com/services/multimedia/Smarter_wireless_networks.pdf

[4] M. Satyanarayanan, et al., "An open ecosystem for mobile-cloud convergence," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 63–70, Mar. 2015.

[5] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, Sep. 2013.

[6] S. Davy, et al., "Challenges to support edge-as-a-service," *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 132–139, Jan. 2014.

[7] Z. Becvar, J. Plachy, and P. Mach, "Path selection using handover in mobile networks with cloud-enabled small cells," in *Proc. IEEE 25th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun.*, Sep. 2014, pp. 1480–1485.

[8] A. Fischer, J. Botero, M. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, Oct.-Dec. 2013.

[9] M. Chowdhury, M. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[10] A. Ksentini, T. Taleb, and M. Chen, "A Markov decision process-based service migration procedure for follow me cloud," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2014, pp. 1350–1354.

[11] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *Proc. IEEE Military Commun. Conf.*, Oct. 2014, pp. 835–840.

[12] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Netw.*, May 2015, pp. 1–9.

[13] M. Srivatsa, R. Ganti, J. Wang, and V. Kolar, "Map matching: Facts and myths," in *Proc. ACM SIGSPATIAL*, 2013, pp. 484–487.

[14] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[15] S. O. Krumke, *Online Optimization: Competitive Analysis and Beyond*. Berlin, Germany: Habilitationsschrift Technische Universitaet Berlin, 2001.

[16] Y. Azar, I. R. Cohen, and D. Panigrahi, "Online covering with convex objectives and applications," *CoRR*, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.3507

[17] N. Buchbinder, S. Chen, A. Gupta, V. Nagarajan, and J. Naor, "Online packing and covering framework with convex objectives," *CoRR*, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.8347

[18] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Evaluation*, vol. 91, pp. 205–228, Sep. 2015.

[19] H.-C. Hsiao, H.-Y. Chung, H. Shen, and Y.-C. Chao, "Load rebalancing for distributed file systems in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 5, pp. 951–962, May2013.

[20] Y. Azar, "On-line load balancing," *Theoretical Comput. Sci.*, vol. 130, pp. 218–225, 1992.

[21] C.-H. Chen, S. D. Wu, and L. Dai, "Ordinal comparison of heuristic algorithms using stochastic optimization," *IEEE Trans. Robot. Autom.*, vol. 15, no. 1, pp. 44–56, Feb. 1999.

[22] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, "On-line routing of virtual circuits with applications to load balancing and machine scheduling," *J. ACM*, vol. 44, no. 3, pp. 486–504, May 1997.

[23] G. Aceto, A. Botta, W. de Donato, and A. Pescape, "Cloud monitoring: A survey," *Comput. Netw.*, vol. 57, no. 9, pp. 2093–2115, 2013.

[24] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1082–1090.

[25] K. LaCurts, J. Mogul, H. Balakrishnan, and Y. Turner, "Cicada: Introducing predictive guarantees for cloud networks," in *Proc. 6th USENIX Conf. Hot Topics Cloud Comput.*, Jun. 2014, pp. 14–14.

[26] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Hoboken, NJ, USA: Wiley, 2007.

[27] B. Korte and J. Vygen, *Combinatorial Optimization*. Berlin, Germany: Springer, 2002.

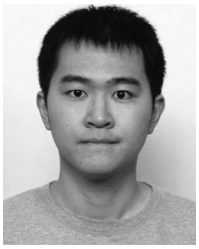
[28] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[29] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *Proc. 1st Int. Commun. Syst. Netw. Workshops*, Jan. 2009, pp. 1–10.

[30] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD data set epfl/mobility (v. 2009–02-24)," Feb. 2009. [Online]. Available: http://crawdad.org/epfl/mobility/.

[31] J. Li, H. Kameda and K. Li, "Optimal dynamic mobility management for PCS networks," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 319–327, Jun. 2000.

[32] J. Li and H. Kameda, "Load balancing problems for multi-class jobs load balancing problems for multi-class jobs in distributed/parallel computer systems," *IEEE Trans. Netw.*, vol. 47, no. 3, pp. 322–332, Mar. 1998.



Shiqiang Wang received the BEng and MEng degrees from Northeastern University, China, in 2009 and 2011, respectively. He received the PhD degree from Imperial College London, United Kingdom, in 2015. He is currently a research staff member with IBM Thomas J. Watson Research Center, United States, where he also worked as a graduate-level co-op in the summers of 2014 and 2013. In the autumn of 2012, he worked with NEC Laboratories Europe, Germany. His research interests include dynamic control mechanisms, optimization algorithms, protocol design and prototyping, with applications to mobile cloud computing, hybrid and heterogeneous networks, ad-hoc networks, and cooperative communications. He has more than 30 scholarly publications, and has served as a technical program committee (TPC) member or reviewer for a number of international journals and conferences. He received the 2015 Chatschik Bisdikian Best Student Paper Award of the Network and Information Sciences International Technology Alliance (ITA).

Rahul Urgaonkar received the bachelor's degree in electrical engineering from Indian Institute of Technology, Bombay, and the master's and PhD degrees in electrical engineering from the University of Southern California. He is an operations research scientist with the Modeling and Optimization Group, Amazon. Previously, he was with IBM Research where he was a task leader on the US Army Research Laboratory (ARL) funded Network Science Collaborative Technology Alliance (NS CTA) program. He was also a primary researcher in the US/UK International Technology Alliance (ITA) research programs. His research is in the area of stochastic optimization, algorithm design and control with applications to communication networks, and cloud-computing systems.

Ting He received the BS degree in computer science from Peking University, China, in 2003, and the PhD degree in electrical and computer engineering from Cornell University, Ithaca, New York, in 2007. She is an Associate Professor in the School of Electrical Engineering and Computer Science at Pennsylvania State University, University Park, PA. From 2007 to 2016, she was a Research Staff Member in the Network Analytics Research Group at IBM T.J. Watson Research Center, Yorktown Heights, NY. Her work is in the broad areas of network modeling and optimization, statistical inference, and information theory. She has served as the membership co-chair of ACM N2Women and the GHC PhD Forum committee. She has served on the TPC of a range of communications and networking conferences, including IEEE INFOCOM (Distinguished TPC Member), IEEE SECON, IEEE WiOpt, IEEE/ACM IWQoS, IEEE MILCOM, IEEE ICNC, IFIP Networking, etc. She received the Research Division Award and the Outstanding Contributor Awards from IBM, in 2016, 2013, and 2009. She received the Most Collaboratively Complete Publications Award by ITA, in 2015, the Best Paper Award at the 2013 International Conference on Distributed Computing Systems (ICDCS), a Best Paper Nomination at the 2013 Internet Measurement Conference (IMC), and the Best Student Paper Award at the 2005 International Conference on Acoustic, Speech and Signal Processing (ICASSP). Her students received the Outstanding Student Paper Award at the 2015 ACM SIGMETRICS and the Best Student Paper Award at the 2013 ITA Annual Fall Meeting. She is a senior member of the IEEE.

Kevin Chan received the BS in ECE/EPP from Carnegie Mellon University, Pittsburgh, Pennsylvania, the MSECE degree and the PhD degree in electrical and computer engineering from Georgia Institute of Technology, Atlanta, GA. He is research scientist with the Computational and Information Sciences Directorate, U.S. Army Research Laboratory (Adelphi, MD). Previously, he was an ORAU postdoctoral research fellow at ARL. His research interests include network science and dynamic distributed computing, with past work in dynamic networks, trust and distributed decision making and quality of information. He has been an active researcher in ARL's collaborative programs, the Network Science Collaborative Technology Alliance and Network and Information Sciences International Technology Alliance.

Murtaza Zafer received the BTech degree in electrical engineering from Indian Institute of Technology, Madras, in 2001, and the PhD and SM degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, in 2003 and 2007, respectively. He currently works with Nyansa Inc., where he heads the analytics portfolio of the company, building a scalable big data system for analyzing network data. Prior to this, he was a senior research engineer with Samsung Research America, where his research focused on machine learning, deep-neural networks, big data, and cloud computing systems. From 2007-2013 he was a research scientist with the IBM Thomas J. Watson Research Center, New York, where his research focused on computer and communication networks, data-analytics, and cloud computing. He was a technical lead on several research projects in the US-UK funded multi-institutional International Technology Alliance program with emphasis on fundamental research in mobile wireless networks. He has previously worked at the Corporate R&D center of Qualcomm Inc. and at Bell Laboratories, Alcatel-Lucent Inc., during the summers of 2003 and 2004, respectively. He serves as an associate editor for the *IEEE Network Magazine*. He is a co-recipient of the Best Paper Award at the IEEE/IFIP International Symposium on Integrated Network Management, 2013, and the Best Student Paper award at the International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt) in 2005. He received the Siemens and Philips Award in 2001 and a recipient of several invention achievement awards at IBM Research.

Kin K. Leung received the BS degree from the Chinese University of Hong Kong, in 1980, and the MS and PhD degrees from the University of California, Los Angeles, in 1982 and 1985, respectively. He joined AT&T Bell Labs, New Jersey, in 1986 and worked at its successors, AT&T Labs and Lucent Technologies Bell Labs, until 2004. Since then, he has been the Tanaka chair professor in the Electrical and Electronic Engineering (EEE), and Computing Departments, Imperial College, in London. He is the head of Communications and Signal Processing Group in the EEE Department. His current research focuses on protocols, optimization and modeling of various wireless networks. He also works on multi-antenna and cross-layer designs for these networks. He received the Distinguished Member of Technical Staff Award from AT&T Bell Labs (1994), and was a co-recipient of the Lanchester Prize Honorable Mention Award (1997). He was elected an IEEE Fellow (2001), received the Royal Society Wolfson Research Merits Award (2004-09) and became a member of Academia Europaea (2012). He also received several best paper awards, including the IEEE PIMRC 2012 and ICDCS 2013. He has actively served on conference committees. He serves as a member (2009-11) and the chairman (2012-15) of the IEEE Fellow Evaluation Committee for Communications Society. He was a guest editor for the IEEE JSAC, IEEE Wireless Communications and the *MONET journal,* and as an editor for the *JSAC: Wireless Series*, the *IEEE Transactions on Wireless Communications,* and the *IEEE Transactions on Communications*. Currently, he is an editor for the ACM Computing Survey and International Journal on Sensor Networks.