Cost Aware Service Placement and Load Dispatching in Mobile Cloud Systems

Lei Yang, Jiannong Cao, *Fellow, IEEE*, Guanqing Liang, and Xu Han

Abstract—With proliferation of smart phones and an increasing number of services provisioned by clouds, it is commonplace for users to request cloud services from their mobile devices. Accessing services directly from the Internet data centers inherently incurs high latency due to long RTTs and possible congestions in WAN. To lower the latency, some researchers propose to 'cache' the services at edge clouds or smart routers in the access network which are closer to end users than the Internet cloud. Although 'caching' is a promising technique, placing the services and dispatching users' requests in a way that can minimize the users' access delay and service providers' cost has not been addressed so far. In this paper, we study the joint optimization of service placement and load dispatching in the mobile cloud systems. We show this problem is unique to both the traditional caching problem in mobile networks and the content distribution problem in content distribution networks. We develop a set of efficient algorithms for service providers to achieve various trade-offs among the average latency of mobile users' requests, and the cost of service providers. Our solution utilizes user's mobility pattern and services access pattern to predict the distribution of user's future requests, and then adapt the service placement and load dispatching online based on the prediction. We conduct extensive trace driven simulations. Results show our solution not only achieves much lower latency than directly accessing service from remote clouds, but also outperforms other classical benchmark algorithms in term of the latency, cost and algorithm running time.

Index Terms—Mobile cloud computing, service placement, load dispatching

1 Introduction

The application of cloud services in the mobile ecosystem enables a newly emerging mobile computing paradigm, namely mobile cloud computing (MCC) [1]. In typical MCC system model, the mobile users directly connect to a centralized cloud on Internet. However, the MCC model that enables the service provisioning directly from large centralized Internet data centers inherently incurs relatively high latency due to long RTTs and possible congestions in WAN [2]. The latency may meet the requirement of some applications like web browsing, but can yield bad usability/user experience for latency sensitive applications such as high quality video streaming, mobile gaming, augmented reality and so on.

To solve the problem, some researchers propose to add another abstraction tier, named as cloudlets [2] or foglets [3], between end devices and traditional Internet data centers. This tier provides computation, storage, and networking resources, aiming to bring the cloud closer to mobile users. The possible places for deploying cloudlets are wireless access networks. The cloudlets could be fixed infrastructures connected to the wireless access networks, or augmented routers and switchers in the wireless access networks. The mobile users normally connect to cloudlets over low latency WLAN, and 3G/4G. Although caching the cloud services on the middle tier cloudlets is a promising technique to reduce the service access latency, how to place the services on the cloudlets and dispatch the users' request load is yet to be addressed.

Given the rapid growth of mobile services and widely deployed cloudlets in the wireless access networks, our primary goal is to help service providers minimize the access latency by serving the mobile users' request load from cloudlets. Meanwhile, we would also like to reduce the cost of service providers which mainly includes the resource usages on cloudlets and the cost of updating the service placements on cloudlets. We explain these goals as follows:

- *Minimize the latency*. User-perceived latency is the most important metric. Considering the geographically distribution of cloudlets, requesting services from various cloudlets and the Internet cloud has different latency. Our goal is to minimize the average latency of all the users' request loads.
- *Minimize the resource usage on cloudlets*. Normally the application providers need to buy the resources from the cloudlets providers to cache the services on cloudlets. From the service providers' perspective, we always aim to use minimal cloudlet resources to achieve best application performance for the mobile users.
- *Minimize the service placement transitions*. We define the service placement transition as the change of placement over time. The placement transition over time needs transmission of associated data with the service among the cloudlets and the cloud. We need to limit the service placement transitions in order to reduce the data transmission cost.

Importantly, we realize that only optimizing one factor in a complex decision process can not satisfy the providers'

• *The authors are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong.*
E-mail: {csleiyang, csjcao, csgliang, csxhan}@comp.polyu.edu.hk.

Manuscript received 11 Jan. 2015; revised 1 Apr. 2015; accepted 29 Apr. 2015. Date of publication 19 May 2015; date of current version 13 Apr. 2016. Recommended for acceptance by E. Kursun.
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/TC.2015.2435781

requirement. What is needed is a way to navigate the three way tradeoff between access latency, resource usage, and service placement transitions.

This is a difficult problem. Although the placement of services on cloudlets and load dispatching can be separately optimized easily in two independent procedures, the optimal decision requires a complex joint optimization of both load dispatching and service placement. Moreover, due to the mobility of users, the request load could vary significantly and frequently in both spatial and temporal domain. The highly dynamic request load implies that it is also necessary to periodically update the decisions, keeping in mind both the current performance and affordable cost, and the expected future workload. These inherent complexities have not been addressed in recent work, which either optimize the service placement and load dispatching separately [10], [11], [12], [16], [17], [18], [19], or study the joint optimization in terms of the traditional data center workloads rather than the highly dynamic mobile workloads [8], [9], [22], [23].

In this paper, we study the joint optimization problem of service placement and load dispatching problem in mobile cloud systems, in particular under highly dynamic mobile workloads. Our major contributions are as follows.

- To the best of our knowledge, this work is the first one to study the service placement and load dispatching problem in mobile cloud systems. We first formulate the snapshot problem, named basic service placement problem (BSPP), which aims to optimize the access latency with the capacity constraints of cloudlets. We show that BSPP is hard to solve and is unique to some classical problems including facility location problem (FLP), data caching in mobile computing, and content distribution in content distribution networks (CDNs).
- We then design a competitive heuristic to BSPP which outperforms a set of benchmark algorithms significantly in terms of the access latency, and algorithm run time.
- We further extend BSPP to a more practical model that aims to optimize the trade-off between access latency, resource usage and service placement transitions, from the standpoint of service providers. We develop an online algorithm for this model that can be deployed directly in practical systems.
- We evaluate our online algorithm using the real world urban transportation dataset and Youtube dataset. The result shows the online algorithm can achieve good performance in both access latency and cost of service providers.

2 System Model and Problem Formulation

2.1 Terminologies

We consider a typical mobile cloud system which contains three tiers: cloud, cloudlets and mobile devices. The mobile users connect with cloudlets through low latency and high bandwidth WLANs. For convenience of description, we first explain the following terminologies which are used throughout of the paper.

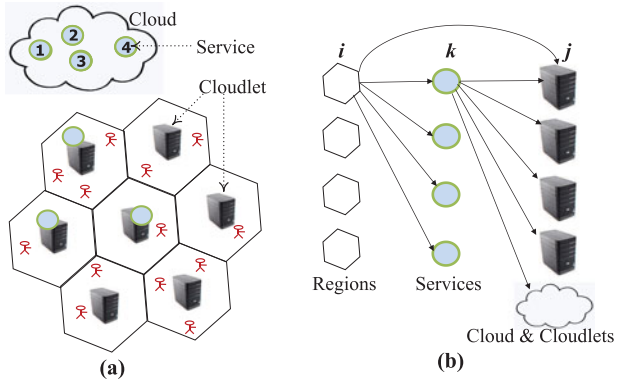


Fig. 1. Illustration of the BSPP.

- *Cloud versus cloudlet*. The *cloud* refers to the centralized data center on Internet, while *cloudlets* are geographically distributed between the cloud and end users. The system includes a sole cloud and multiple cloudlets. We also name the cloud and the cloudlet as *resource node*.
- *Service (or data)*. Service is an abstraction of applications that is hosted by the resource node and requested by mobile users. It can be video streaming, social gaming, navigation, reality augmented and so on. Service is usually associated with data. Placing services on cloudlets needs to store associated data on them. Thus, we use the terminology *service* and *data* interchangeably.
- *Load (or demand)*. Load or demand is the number of mobile users who request the service. The services often have different loads. The load of service is geographically distributed, and varies with time.
- *Service (or data) placement*. Service placement is to decide, for each service, which cloudlet it is placed at. Each service is allowed to be placed on multiple cloudlets. All the services have been persistently placed on the cloud in advance.
- *Load dispatching*. Load dispatching is to distribute the load onto cloudlets and the cloud.

2.2 Basic Service Placement Problem

Fig. 1 illustrates the system model. We consider a 2-D area deployed with a set of cloudlets. The number of cloudlets is N. Each cloudlet has a coverage area or region. We assume that the 2-D area is completely covered by all the N cloudlets, and the cloudlets' coverage regions do not have overlap with each other. Thus, we can divide the whole 2-D area into N regions respectively covered by the N cloudlets. We use i to index the region, and j to index the cloudlet. Both i and j satisfy $i \in [1, N]$ and $j \in [1, N]$. Suppose the service provider has a set of K services to place on the cloudlets. Let k be the index of each service, $1 \leq k \leq K$. We use n_i^k to denote the reginal load, which indicates the number of users from region i that request the service k. Table 1 shows the mathematical notations in this paper.

We define a $N \times (N+1)$ matrix $[d_{ij}]$ to represent the latency of load from region i to services on cloudlets or the cloud j. For convenience, we allow $j = N+1$, which indicates the cloud on Internet, while $1 \leq j \leq N$ represents the cloudlets. Note that $[d_{ij}]$ are constants with the properties:

TABLE 1
Mathematical Notations in This Paper

i	index of the region;
j	index of the resource node (cloud and cloudlets);
k	index of the service;
N	the number of regions (and cloudlets);
n_i^k	the number of users from region i that request the service k;
d_{ij}	the latency of accessing services on cloudlet j from region i ;
α_j^k	a binary variable α_j^k that represents whether service k is placed on cloudlet j;
β_{ij}^k	a continuous variable ranging from 0 to 1 that represents the portion of regional load n_i^k to be dispatched to and served by the cloudlet/cloud j;
Λ_k	the size of the associated data with service k;
u_j	the storage capacity of cloudlet j;
c_j	the maximum load that cloudlet j can serve;
μ	the constant to normalize the resource usage cost;
γ	the constant to normalize the service placement transition cost;

$\forall i, d_{ij}$ has the least value if $j = i$, while has the greatest value if $j = N + 1$. The property indicates that if the user requests services on the cloudlet in its proximity, the latency is the lowest. If the user requests services on the cloudlets far away, the latency increases. The worst case is that the user requests the service from the Internet cloud, where the latency is greatest.

Definition 1. (Basic Service Placement Problem). *Given the load from each region, the problem is to decide, for each cloudlet, which services it should be placed with, and decide for each user's request, which cloudlet it should be dispatched to and served at, such that the average latency of all the requests is minimized while the capacity constraint of each cloudlet is satisfied.*

Definition 1 describes the basic server placement problem. We now formulate the problem in Equation (1). The problem has two types of decision variables. The first one is a binary variable α_j^k that represents whether service k is placed on cloudlet j, where $j \in [1, N]$. The second one is a continuous variable β_{ij}^k that represents the portion of regional load n_i^k to be dispatched to and served by the cloudlet/cloud j, where $j \in [1, N+1]$, and $\forall i$, $\beta_{ij}^k \in (0, 1)$, $\sum_j \beta_{ij}^k = 1$.

Next, we model the capacity constraints of cloudlets. The cloudlet is constrained in its storage and computation capability. Due to constrained storage, each cloudlet is able to be placed with limited number of services. We denote the size of the associated data of service k by Λ_k. Placing service k on the cloudlet requires Λ_k storage resources. We define a constant vector $[u_j]$ to represent the storage capacity of each cloudlet j. Due to constrained computation capability, each cloudlet is able to serve limited amount of load. Thus, we define a constant vector $[c_j]$ to represent the maximum load that each cloudlet j can serve. For simplicity, we assume the computation resources needed to serve one request is the same for all the services. We formulate the BSPP by

$$\min_{\alpha_j^k, \beta_{ij}^k} \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{N+1} \beta_{ij}^k \times n_i^k \times d_{ij} \tag{1}$$

s.t.

$$\beta_{ij}^k \leq \alpha_j^k, \forall i, \forall j \in [1, N+1], \forall k, \tag{2}$$

$$\sum_{k=1}^{K} \Lambda_k \times \alpha_j^k \leq u_j, \forall j \in [1, N], \tag{3}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{N} \beta_{ij}^k \times n_i^k \leq c_j, \forall j \in [1, N], \tag{4}$$

$$\sum_{j=1}^{N+1} \beta_{ij}^k = 1, \forall i, \forall k, \tag{5}$$

$$\beta_{ij}^k \geq 0, \forall i, \forall j \in [1, N+1], \forall k, \tag{6}$$

$$\alpha_j^k = 0 \text{ or } 1, \forall j \in [1, N+1], \forall k. \tag{7}$$

The BSPP aims to minimize the average latency of all the loads. Equation (2) indicates that the load can only be dispatched to the cloudlet with the requesting services placed. Equations (3) and (4) respectively present the storage and computation capacity constraints.

2.3 Cost-Aware Service Placement Problem

We now consider a more practical problem, called cost-aware service placement problem (CSPP). In this problem, we take into account the cost of the service provider. It includes two parts: the cost of resource usage on the cloud and cloudlets, and the cost caused by the transition of service placements on cloudlets over time. The cost of service providers and the average latency of users' requests are two conflicting objectives. Placing more replicas of services on cloudlets can always lower the average latency of the loads, meanwhile it increases the resource usage on cloudlets. CSPP aims to achieve the best trade-off between them.

• *Cost of resource usage*. The cost of resource usage arises from storage and computation cost on cloudlets, and computation cost on the cloud. To simplify the problem, we assume that the storage cost per unit of data on the resource nodes are the same. The computation cost per request for various services are the same on the resource nodes. We assume that the services have been persistently placed at the cloud in advance. Therefore, the total resource cost varies depending on the service placement α_j^k on the cloudlets.

• *Cost of service placement transition over time*. We consider the time dimension, and study the service placement during certain period in which the loads from multiple regions may vary. We take into account the extra cost caused by service placement transitions over two successive time slots. The service placement transition requires transmissions of the associated data with services that are added onto the cloudlets. The transmissions cause a lot of bandwidth cost between the cloud and cloudlets, or among cloudlets. Therefore, from the perspective of application provider, we

need to restrict service placement transitions in order to the total lower operational cost.

Definition 2. (Cost-Aware Service Placement Problem). *Given regional loads at each time t during one certain period, the problem is to decide at each time slot, for each cloudlet, which services it should be placed with, and for each user's request, which cloudlet it should be dispatched to and served at, such that the weighted summation of latency, resource usage cost and service placement transition cost over the whole period is minimized, while the capacity constraint of each cloudlet is satisfied.*

• *Formulation of CSPP.* Let t be indices of time slots. The duration of one time slot can range from several minutes to a few hours. It depends on the frequency that the regional load changes over time. In real world systems, the frequency is affected by a few factors such as the spatial density of cloudlets in the 2-D whole area, moving speed of mobile users, and users' service accessing pattern. We will model how the factors affect the load variance, and describe our load prediction method in Section 4. We assign the notations in Problem (1) with the time indices t. For example, $n_i^k(t)$ indicates the load from region i for service k at time t, and $\alpha_j^k(t)$ and $\beta_{ij}^k(t)$ are the decision variables at time t. Then, we formulate CSPP by

$$\begin{aligned}\min_{\alpha_j^k(t),\beta_{ij}^k(t)} \quad & \sum_t \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{N+1} \beta_{ij}^k(t) \times n_i^k(t) \times d_{ij} \\ & + \mu \cdot \sum_t \sum_{k=1}^{K} \sum_{j=1}^{N} \alpha_j^k(t) \times \Lambda_k \\ & + \gamma \sum_t \sum_{k=1}^{K} \sum_{j=1}^{N} \mathbb{I}[\alpha_j^k(t) - \alpha_j^k(t-1)] \times \Lambda_k, \\ \text{s.t.} \quad & \forall t, (2), (3), (4), (5),\end{aligned} \tag{8}$$

where $\mathbb{I}[\cdot]$ is a function, and $\mathbb{I}[x] = 1$ if $x > 0$, otherwise $\mathbb{I}[x] = 0$ for $x \leq 0$. Note that placement transition cost increases with the size of the data associated with the service, since the cost is due to the data transmission of the data associated with the newly placed service.

μ and γ are constants set by Equations (9) and (10). In the equations, $\sum_i n_i^k d_{i,N+1}$ represents the maximal latency when all the loads are served by the cloud. $\sum_{j=1}^{N} u_j$ represents the maximal service placement cost when the cloudlets are fully placed with the services. These two are used to respectively normalize the latency and the cost

$$\mu = \frac{\sum_{k=1}^{K} \sum_{i=1}^{N} (n_i^k \cdot d_{i,N+1})}{\sum_{j=1}^{N} u_j} \times \frac{\Gamma_2}{\Gamma_1}, \tag{9}$$

$$\gamma = \frac{\sum_{k=1}^{K} \sum_{i=1}^{N} (n_i^k \cdot d_{i,N+1})}{\sum_{j=1}^{N} u_j} \times \frac{\Gamma_3}{\Gamma_1}. \tag{10}$$

We can select different trade-offs among the three objectives, i.e., latency, resource usage cost and service placement transition cost, through setting three none-negative preference values Γ_1, Γ_2 and Γ_3, where $\Gamma_1 + \Gamma_2 + \Gamma_3 = 1$.

2.4 Uniqueness of the Problem

We first compare our problem to the classical Facility Location Problem [6], [7], which is very similar to BSPP shown in Equation (1). In FLP, there is a number of candidate locations where the facility can be set up. We have a number of markets, each of which has certain amount of demands that are supposed to be transported from the facilities. The problem determines the locations to set up the facilities and as well dispatches the demands to the facilities being set up, aiming to minimize the sum of the setup cost of facilities and the transportation cost of serving the demands. The transportation cost is determined by the distance between the markets and facilities. If each facility has capacity constraint, being able to serve a limited number of demands, we name the problem as capacitated FLP.

Intuitively, we can model BSPP as similar to capacitated FLP as possible. Assume we have $N \times K$ markets, and each market's demand is n_i^k. There are $N \times K$ locations to set up the facilities. The 0-1 binary variable α_j^k represents whether the facility is set up at location (j, k). The distance between the markets and candidate locations of facilities is measured by latency. Note that the distance between market (i, k) and location (j, k) is d_{ij} if the market and location has the same k; otherwise, the distance is infinite.

The key difference between BSPP and FLP is the capacity constraint. In FLP, each location if being placed with the facility has certain capacity to serve the demand. In BSPP, we constrict the total capacity of the set of K locations (j, k) having the same j. It is due to the computation capacity c_j of each cloudlet. Furthermore, BSPP restricts that only a limited number can be selected to set up the facilities from the K locations (j, k) having the same j, because of the storage capacity u_j of each cloudlet. Due to the complicate constraints, BSPP is different from and more difficult then FLP.

This problem is also unique to tradition cache placement problems in mobile networks, and content distribution in CDNs. The details are explained in Section 6.

3 Solutions to BSPP

3.1 Problem Transformation

The idea of our solution is to relax the constraints of cloudlets storage capacities in Equation (3). Assume that each cloudlet has unlimited storage capacity, and can be placed with all the services. The problem is to assign the regional loads to the resource nodes, such that the average latency of all the users' requests is minimized. The BSPP with the relaxation of cloudlet storage capacities is then equal to the transportation problem [24]. The N regions in BSPP can be considered as N destinations. The resource nodes are equal to sources or factories. Each destination i has a demand of $\sum_k n_i^k$ units. Each source j can supply c_j units. The transportation cost from source j to destination i is the matrix $[d_{ij}]$. The problem is to determine which sources should supply which destinations so as to minimize the transportation cost.

We can obtain optimal solution of transportation problem by using linear programming. We denote the optimal solution as $[x_{ij}^*]$. x_{ij}^* indicates the amount of load from region i that are dispatched to and served by resource node j. Next, given the load sequence $[n_i^1, n_i^2, \ldots, n_i^k, \ldots]$ for various services from region i, and the amount of load x_{ij}^* assigned to

resource node j, we need to determine corresponding sequence $[\beta_{ij}^1, \beta_{ij}^2, \ldots, \beta_{ij}^k, \ldots]$ such that $\sum_k n_i^k \beta_{ij}^k = x_{ij}^*$. We name $n_i^k \times \beta_{ij}^k$ as the contribution from service k to the assignment x_{ij}^*. The problem of determining contributions from various services to the given assignment x_{ij}^* is formulated as solving the Equations (11)-(13). We can find infinite number of feasible solution β_{ij}^k satisfying these Equations (11)-(13):

$$\sum_k n_i^k \beta_{ij}^k = x_{ij}^*, \forall i, \forall j \in [1, N+1], \tag{11}$$

$$\sum_{j=1}^{N+1} \beta_{ij}^k = 1, \forall i, \forall k, \tag{12}$$

$$\beta_{ij}^k \geq 0, \forall i, \forall j \in [1, N+1], \forall k. \tag{13}$$

We now consider the constraints of the cloudlet storage capacity by adding three more constraints shown in Equations (2), (3), (7). We may not find feasible β_{ij}^k by solving the set of Equations (2), (3), (7), (12), (13). However, we can always find β_{ij}^k and α_j^k such that $\sum_k n_i^k \beta_{ij}^k$ approaches x_{ij}^* as much as possible. Let $\sum_k n_i^k \beta_{ij}^k = x_{ij}$. The original problem can be transferred into:

$$\begin{aligned} &\min \sum_i \sum_j |x_{ij} - x_{ij}^*|^2, \\ &\text{s.t.} \quad (2),\ (3),\ (7),\ (12),\ (13). \end{aligned} \tag{14}$$

We explain the reason why we want x_{ij} to approach x_{ij}^* as much as possible. The relaxed problem is linear. In its solution space, the lower distance the solution $[x_{ij}]$ is located from the optimum $[x_{ij}^*]$, the less the average latency is.

3.2 Greedy Heuristic

Through the problem transformation, we design a simple yet efficient heuristic to solve BSPP as shown in Algorithm 1. The key part of the algorithm is to determine service placement and load dispatching, such that the actual $[x_{ij}]$ approximates to $[x_{ij}^*]$ as much as possible (Lines 1-21). We describe this part at first.

3.2.1 Approximating $[x_{ij}]$ to $[x_{ij}^*]$

After obtaining the optimum x_{ij}^* (Line 1), let us think in the reverse direction to the original demand-and-supply problem. Consider the N regions as factories or suppliers. Consider the $N+1$ source nodes as $N+1$ destinations or customers. Each factory i stores K types of goods. The amount of k-th type of goods stored at factory i is n_i^k. Each destination j demands x_{ij}^* units of goods from the factory i. Each destination j can accept a limited number of goods types. Our objective is to transport the goods from factories to destinations such that we satisfy as much demand x_{ij}^* as possible.

Our algorithm contains two steps. The first step is destination selection. The destination that demands more units of goods is selected earlier (Line 3). The second step is to transport the demanding goods from factories to the selected destination. In this step, for each selected destination j, we check each type of stored goods at the factories, and calculate how much of the total demand $\sum_i x(i,j)$ by destination j can be supplied if this type of good is accepted by the destination (Line 12). The type of goods that can supply maximum demand by destination i is selected to be accepted by destination i (Line 14). We select the acceptable types of goods for destination j iteratively until the constraint of the acceptable types of goods can not be satisfied, or the remainder demand $\sum_i x(i,j)$ by destination j is zero (Lines 7-20).

Algorithm 1. The Greedy Heuristic for BSPP

Input: n_i^k, u_j, and c_j
Output: α_j^k, β_{ij}^k

1 Obtaining the optimal assignment $[x_{ij}^*]$ by solving the relaxed linear programming in Equation (1);
2 Initialize $\rho(i,j,k) \leftarrow 0$, $\phi(i,k) \leftarrow n_i^k$, $\sigma(i,j) \leftarrow x_{ij}^*$, $\alpha_j^k \leftarrow 0$, $\forall i, j, k$;
3 Construct the list L_A of source nodes j according to the descending order of $\mathcal{D}(j) = \sum_i x_{ij}^*$;
4 **while** *the list L_A is not empty* **do**
5 Select the first source node j from the list L_A;
6 Initialize the set S_j of services which are placed at source node j;
7 **while** *there exists none-zero element in the set* $\{\sigma(i,j)|\forall i\}$ **do**
8 Construct the list L_{CS} of candidate services that can be placed on source node j, and initialize $L_{CS} \leftarrow \emptyset$;
9 **for** *each service* $k' \notin S_j$ **do**
10 **if** $\Lambda_k + \sum_{k \in S_j} \alpha_j^k \cdot \Lambda_k \leq u_j$ **then**
11 Add service k' into the L_{CS};
12 $\mathcal{P}(k') = \sum_{i=1}^{N} \min\{\phi(i,k'), \sigma(i,j)\}$;
13 **if** L_{CS} *is not empty* **then**
14 Place the service k_0 from L_{CS} that has maximum $\mathcal{P}(k)$ onto source node j;
15 **for** *each region* i **do**
16 $\rho(i,j,k_0) \leftarrow \min\{\phi(i,k_0), \sigma(i,j)\}$;
17 $\phi(i,k_0) \leftarrow \phi(i,k_0) - \rho(i,j,k_0)$;
18 $\sigma(i,j) \leftarrow \sigma(i,j) - \rho(i,j,k_0)$;
19 **else**
20 Break;
21 Remove the source node j from the list L_A;
22 **for** *each pair* (i,k), $i \in [1,N], k \in [1,K]$ **do**
23 **if** $\phi(i,k) > 0$ **then**
24 Initialize the list L_B of the cloudlets with service k placed: $L_B \leftarrow \{j|\alpha_j^k = 1, j \in [1,N]\}$;
25 Sort L_B by the ascending order of d_{ij};
26 **while** L_B *is not empty, and* $\phi(i,k) > 0$ **do**
27 Select the first source node j in L_B;
28 $\mathcal{F}(j) = \min\{c_j - \sum_{i,k} \rho(i,j,k), \phi(i,k)\}$;
29 $\phi(i,k) \leftarrow \phi(i,k) - \mathcal{F}(j)$;
30 $\rho(i,j,k) \leftarrow \rho(i,j,k) + \mathcal{F}(j)$;
31 Remove j from list L_B;
32 **if** $\phi(i,k) > 0$ **then**
33 $\rho(i,N+1,k) \leftarrow \rho(i,N+1,k) + \phi(i,k)$;
34 $\phi(i,k) \leftarrow 0$;
35 $\beta_{ij}^k \leftarrow \rho(i,j,k)/n_i^k, \forall i, \forall j, \forall k$;
36 **return** α_j^k and β_{ij}^k;

3.2.2 Load Assignment of Non-zero $[x_{ij}^* - x_{ij}]$

For the non-zero elements of $\phi(i,k)$, we need to assign them to the destinations (Lines 22-34). Our approach is to assign the remaindering goods (or loads) $\phi(i,k)$ to the destinations $\{j|1 \leq j \leq N\}$ (or the cloudlets) which satisfy two conditions: 1) the destination can accept the type k of goods, i.e., $\alpha_j^k = 1$; 2) the destination has remainder demand that are not

supplied, i.e., $\sum_i \sigma(i,j) > 0$. We call the destinations meeting both the conditions as candidate destinations. We sort the candidate destinations by the ascending order of their transportation cost d_{ij} to the factory. We assign $\phi(i,k)$ with higher priority to the candidate destination with less transportation cost d_{ij}. After the assignment, if the remainder goods (or loads) are none zero, we assign it to the destination $N+1$ (or the cloud). The destination $N+1$ (or the cloud) does not have capacity constraint, and usually has maximal transportation cost to the factory (or latency to the region).

3.3 Benchmark Algorithms

3.3.1 Linear Programming Based Heuristic

The idea of LP-based heuristic is to relax the integer constraints, and solve the relaxed standard LP problem. Then, we will progressively set the non-integer variables to be 0-1 integers. Specifically, the LP-based heuristic for solving BSPP contains the following steps.

Step 1. Relax the integer constraints of all the $N \times K$ α_j^k, through replacing the constraint (7) with

$$0 \leq \alpha_j^k \leq 1, \forall j,k. \tag{15}$$

Solve the standard LP, and obtain the $N \times K$ α_j^ks.

Step 2. Select one from all the none-integer α_j^ks, and set it to be 1. Make sure the constraint (3) is still satisfied after the setting. Solve the LP with constraints (2), (4), (5), (6) given the modified α_j^k, and obtain an objective value. If there exists multiple selections from the none-integer α_j^ks which still satisfy the constraint (4), we will choose the one to set as 1 which has lowest objective value. If we can not select any one to be set as 1, then we will change to select one from all the none-integer α_j^k and set it to be zero. We always choose the one that results in lowest objective function to set as zero. In other words, in this step, we prefer to set one of all the none-integer α_j^k to be 1. If none can be set as 1, then we choose to select the best one and set it as zero.

Step 3. Repeat the step 3 until all the none-integer α_j^k are set to be integers.

In the LP-based heuristic, the Step 2 will be repeated for $N \times K$ times. In Step 2, solving the standard LP given the α_j^k will be repeated for at most $2 \times N \times K$ times. Each time of LP solving has complexity of $O(n^3)$ if the LP solver in [4] is used, where n is the number of the variables in LP. Here, the variables of the LP are β_{ij}^k, thus we have $n = N \times (N+1) \times K$. The complexity of the LP-based heuristic is $O(N^5K^5(N+1)^3)$.

3.3.2 FLP Based Heuristic

Although FLP is different from BSPP, we are interested to know how the heuristic based on existing solution to FLP performs when it is used to solve the BSPP. Section 2.4 shows the we can formulate BSPP in a very similar way to FLP. In the formulation, we have $N \times K$ markets. Each market (i,k) has demand n_i^k. We have $N \times K$ locations to set up the facility. Setting up facility at location (j,k) has the cost of Λ_k. The key difference between FLP and BSPP is that: 1) In BSPP we classify facilities into N groups. The facilities that are set up at the locations (j,k) with the same j are grouped together. BSPP has capacity constraint for each group, while FLP has capacity constraint for each single facility. 2) BSPP has one additional constraint. For each group, the total cost of setting up the facilities can not exceed the cloudlet's storage capacity. The FLP based heuristic is as follows.

Step 1. Remove the two constraints described above, and obtain an initial solution using the approximate algorithm to uncapacitated FLP [6].

Step 2. For each group of facilities, we check whether the total cost in setting up facilities violates the storage constraint u_j. If it is violated, choose one facility from the group, and close it. The demands served by the closed facility are assigned to others in the group.

Step 3. For each group of facilities, we check whether their total capacity c_j has been exceeded by the assigned demands. If it is, move some demands to other groups who still have margin.

Note that the selection of closed facility and the demand assignment in Step 2, and the movement of demands in Step 3 are done in the way such that the increase of transportation cost due to these operations is minimized.

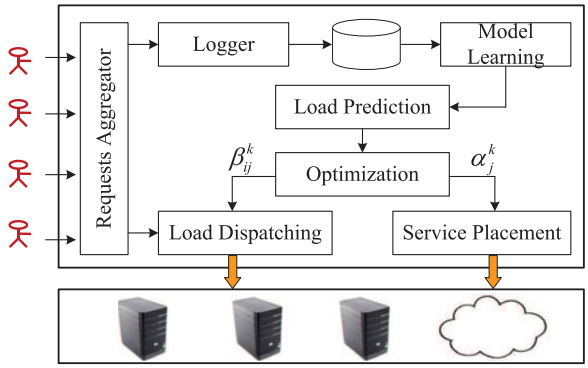


Fig. 2. Overview of the online solution to CSPP.

3.3.3 Genetic Algorithm

In this section, we use genetic algorithms to solve the BSPP. The key tasks for using genetic algorithm are to determine the chromosome representation and the corresponding fitness function to evaluate the chromosomes. We represent the chromosome using the 0-1 service placement variables α_j^k. Thus, the chromosome contains $N \times K$ bits. Given the service placement α_j^k, if the service placement satisfies the cloudlet storage constraints, we can easily find the optimal load dispatching β_{ij}^k by solving a standard linear programming (LP) problem. The LP problem has objective function in Equation (1) and subjects to Equations (2), (4), (5), (6). For any chromosome, if it does not satisfy the cloudlet storage constraint, we assign its fitness with zero; otherwise, we directly assign its fitness with optimum by solving the LP problem.

4 Online Solution to CSPP

4.1 Overview of the Online Solution

In this section, we focus on the design of an online solution to CSPP that can be deployed and applied for service placement and load dispatching in practical systems. Fig. 2 shows the overview of the online solution. We now

illustrate the key components and how they work together in real world deployment.

The system contains three essential components: request aggregate, load prediction, and optimization, which are implemented on the cloud. The *request aggregate* receives mobile users' requests for accessing services, and redirect the requests to the resource nodes that are assigned to serve those requests. The system records three important items of each request, i.e., 1) when it is sent to the cloud, 2) where it is sent from; 3) which service it is to access. The records are stored in the database over time, and used to learn the load prediction model in off-line phases. With the learned model, the system can predict online the load distribution in future time slots based on the most updated load distribution at the current time. After that, the system makes the load dispatching decisions β_{ij}^k and service placement decisions α_j^k by solving the optimization Problem (8). In the following, we present the load prediction, and the optimization with details.

4.2 Prediction of Load Distribution

We describe our approach to predict the load $n_i^k(t)$ at time slot t based on the historical loads $n_i^k(1), n_i^k(2), \ldots, n_i^k(t-1)$. For convenience, we first introduce a few new notations. Let l_t denote the region where the user is located at time t. $\mathbb{L}$ represents the location domain, where $\mathbb{L} = \{L_1, L_2, \ldots, L_N\}$. Note that L_i represents the i-th region. Let s_t denote the service that the user requests at time t. $\mathbb{S}$ denotes the service domain, where $\mathbb{S} = \{S_0, S_1, S_2, \ldots, S_K\}$. $S_1, S_1, \ldots, S_K$ represents the K services that users can request, while S_0 represents the user does not request any service.

At each time slot, the user is located at one of the N regions. The user either requests one of the K services, or request no service. The load prediction is built on the joint Transition Probability $P(L_i, S_k | L_{i'}, S_{k'})$ of user's Location and Service accessing pattern. We name it as TPLS. If we know the TPLS, we can predict the load at time t by

$$n_i^k(t) = \sum_{i'} \sum_{k'} n_{i'}^{k'}(t-1) \times P(L_i, S_k | L_{i'}, S_{k'}). \tag{16}$$

In the following, we introduce our method to estimate the TPLS. Intuitively, we can estimate the probabilities through the relative frequency of each transition from (L_i, S_k) to $(L_{i'}, S_{k'})$ by observing the loads in past time slots. However, the number of parameters is too large. Accurate estimation requires too many load samples.

We now simplify the estimation of TPLS according to the following three observations. First, user's locations evolve over time according to certain mobility pattern. Second, the service that the user accesses at time slot t has certain dependence with the services he/she accesses during past time slots. We name it service accessing pattern. Third, the service that the user accesses has certain dependence with his/her location. From these observations, we model the predictor using the conditional probability equation

$$P(s_t, l_t | s_{t-1}, l_{t-1}) = P(s_t | s_{t-1}, l_t) \cdot P(l_t | l_{t-1}). \tag{17}$$

We use Malcov process to model the evolution of l_t and s_t. Thus, we can obtain $P(l_t | l_{t-1})$ easily through a $N \times N$ location transition probability matrix (LTPM). Each element in LTPM indicates the transition probability between the location L_i and $L_{i'}$. We estimate $P(s_t | s_{t-1}, l_t)$ through a number N of $(K+1) \times (K+1)$ service transition probability matrixes (STPMs), as the service transition probabilities depend on the user's location. Now the prediction model parameters consist of one LTPM and N STPMs. We have deployed a backend agent to record the user's location and corresponding accessing service whenever one user initiates an service accessing request. At the beginning of each time t, we calculate or update the model parameters, i.e., LTPM and STPMs, using the records at time $t-1$. The transition probabilities in LTPM and STPMs are estimated by relative frequency. With the model parameters, we calculate the TPLS $P(L_i, S_k | L_{i'}, S_{k'})$, and then estimate the load distribution at time t by Equation (16).

4.3 Solving One Shot Optimization

Given the service placement $\alpha_j^k(t-1)$ at time $t-1$ and the predicted load distribution $n_j^k(t)$ at time t, the one shot optimization of CSPP is to determine the service placement and load dispatching at time t such that weighted summation of the average latency, resources cost and service placement switching cost is minimized. In this section, we explain how to adapt the greedy heuristic of BSPP to solve the one shot CSPP. The greedy heuristic for CSPP contains three phases. In the first phase, we solve the relaxed transportation problem in Equation (1). In the second phase, after obtaining the optimal solution $[x_{ij}^k]$, we determine the service placement under a greedy policy. In BSPP, the policy is to select the service iteratively that can supply as much load to the demand as possible. However, in CSPP, the policy is to select the one from all the $N \times K$ service placements, which has maximal utility. The utility function is defined by

$$\begin{aligned} \mathcal{U}(j,k) = \sum_{i=1}^{N} [\min\{\phi(i,k), \sigma(i,j)\} \times (d_{i,N+1} - d_{i,j})] \\ -\mu \times \Lambda_k - \gamma \times \Lambda_k \times \left[1 - \alpha_j^k(t-1)\right], \end{aligned} \tag{18}$$

The equation includes three items. Assuming that at the beginning all the loads are assigned to the cloud, the first item in the utility function $\mathcal{U}(j,k)$ represents how much latency can be reduced by adapting the loads requesting service k from the cloud to the cloudlet j. The second item represents the resource usage cost of placing service k onto the cloudlet j. The third item represents the transition cost from the previous service placement. The utility function indicates to what extent the placement of service k onto cloudlet j can increase the objective function of CSPP. The iterative procedure of service placement is terminated until no service placement has positive utility. In the third phase, with the service placement α_j^k, we assign the remainder load $\phi(i,k)$ to the resource nodes. Algorithm 2 describes the pseudo-code of the greedy heuristic for CSPP. In Algorithm 3, the notations being not indicated with time index represent the variables at time slot t, while notations for time slot $t-1$ are explicitly added with the time index. The first phase and third phase in the heuristic algorithm for CSPP are the same with the heuristic in Algorithm 1. Line 3-16 shows the different greedy policy with BSPP to determine the service placement. The complexity of Algorithm 2 is $O(K^2N^2)$.

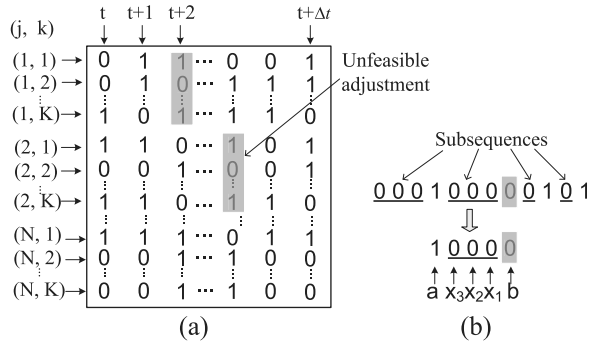


Fig. 3. The illustration of Δt-steps look ahead.

Algorithm 2. The Heuristic for One Shot Optimization

Input: n_j^k, u_j, c_j
Output: α_j^k, β_{ij}^k

Obtaining the optimal assignment $[x_{ij}^*]$ by solving the relax linear programming in Equation (1);
Initialize $\rho(i,j,k) \leftarrow 0$, $\phi(i,k) \leftarrow n_i^k$, $\sigma(i,j) \leftarrow x_{ij}^*$, $\alpha_j^k \leftarrow 0$, $\forall i,j,k$;
Construct a set S_P of all the candidate service placements: $S_P \leftarrow \{(j,k)|1 \le j \le N, 1 \le k \le K\}$;
while $S_P \neq \emptyset$ **do**
  for *each element* (j,k) *in the set* S_P **do**
    Calculate the utility of placing service k onto cloudlet j using Equation (18);
  Initialize the list L_C by $L_C \leftarrow \emptyset$;
  Select the elements (j_0,k_0) in S_P that satisfy: i) $\mathcal{U}(j_0,k_0) > 0$; 2) $\sum_{k=1}^{K} \alpha_{j_0}^k \cdot \Lambda_k + \Lambda_{k_0} \le u_{j_0}$;
  and add the elements into L_C by the descending order of their utilities;
  if $L_C = \emptyset$ **then**
    Break;
  Select the first element (j_1,k_1) in L_C, and place the corresponding service k_1 on the cloudlet j_1;
  for *each region* $i \in [1,N]$ **do**
    $\rho(i,j_1,k_1) \leftarrow \min\{\phi(i,k_1), \sigma(i,j_1)\}$;
    $\phi(i,k_1) \leftarrow \phi(i,k_1) - \rho(i,j_1,k_1)$;
    $\sigma(i,j_1) \leftarrow \sigma(i,j_1) - \rho(i,j_1,k_1)$;
  Remove the element (j_1,k_1) from the set S_P;
Repeat Line 22-35 in Algorithm 1;
return α_j^k, β_{ij}^k;

4.4 Online Algorithm with Δt-Step Look Ahead

The one shot TSCPP predicts the load distribution in the next time slot, and aims to optimize the service placement according to the placement in the previous time slot. Now we are interested in the performance, if we predict the load distribution in the following Δt time slots and optimize the service placement over the Δt time slots. In this section, we design an online algorithm with Δt steps look ahead.

The idea of the online algorithm is as follows. First, we determine the service placement at each one of the future Δt time slots by using the heuristic algorithm for one shot CSPP. Then, we adjust the service placement towards the Δt-step look ahead optimum. To describe the adjustment mechanism, we introduce two terms. We define $\mathcal{G}(j,k)$ as the *gain* of placement (j,k). It is the decrease on latency caused by the placement (j,k) minus its resource storage cost, which is equal to the first two items in Equation (18). We define $\mathcal{C}(j,k)$ as the switching cost caused by placement (j,k), which is equal to the third item in Equation (18). Therefore, we have the utility $\mathcal{U}(j,k) = \mathcal{G}(j,k) - \mathcal{C}(j,k)$.

The adjustment to the one shot solution aims to increase the total utility over the entire period from $t+1$ to $t+\Delta t$. In the one shot solution, if one service k_0 is placed on resource node j_0 at time $t+1$, i.e., $\alpha_{j_0}^{k_0}(t+1) = 1$, we will not adjust the service placement. Oppositely, if one service k_0 is not placed on the resource node j_0, i.e., $\alpha_{j_0}^{k_0}(t+1) = 0$, we need to be cautious to adjust this placement (j_0,k_0). We will check the decisions on the same placement (j_0,k_0) in the following time slots, and pursue the best adjustment that can bring as much increase as possible to the utility over the entire period.

Algorithm 3 shows our method to adjust the one shot solution. We put the service placements into a matrix as shown in Fig. 3, in which the column represents the placement at one time slot. Each column contains $N \times K$ 0-1 placement decisions. The raw reflects the decisions for the same placement (j,k) in the next Δt time slots. Given the placement decisions at time t, we will adjust the placement at $t+1, t+2, \ldots$, and $t+\Delta t$. We do the adjustment raw by raw. In each raw, we distinguish the elements that can not be adjusted. These elements either have none zero values, or can not be set as one due to the storage constraint of the corresponding resource node (see the gray mask in Fig. 3 as an example). Consider the raw as a binary sequence including 0-1 elements, and the unadjustable elements as isolators. The sequence is then divided into multiple subsequences by the unadjustable elements. We treat each subsequence independently, and adjust the subsequence to increase as much utility as possible over all the time slots that the subsequence spans.

Algorithm 3. Δt-Steps Look Ahead Algorithm

Input: The matrix in Fig. 3a, including the previous placement $\alpha_j^k(t)$, and the one shot TSCPP solutions, $\alpha_j^k(t+1), \ldots, \alpha_j^k(t+\Delta t)$
Output: the Δt-steps look ahead solution

for *each raw in the matrix of* Fig. 3a **do**
  Search the feasible subsequences of consequence zeros (see Fig. 3a);
  while *select the subsequence according to the time order* **do**
    Adjust the selected subsequence;
    for *each element* x_δ *in the selected subsequence that has been adjusted as 1* **do**
      Update the gain $\mathcal{G}(\cdot)$ of all the elements in the same column with x_δ (see Line 5 in Algorithm 2);
return *the adjusted* matrix

5 Performance Evaluation

5.1 Evaluation of BSPP

In this section, we evaluate the performance of various algorithms for BSPP through large scale test cases. Table 2 shows the default values of the parameters used in all the experiments. In the sensitivity tests, we change the value of one of these parameters and set other parameters as the default values. In each parameter setting, we randomly generate 2,000 test cases. These random test cases are mainly different in the load distribution $[n_i^k]_{N \times K}$.

TABLE 2
Experiment Parameters

Number of regions (or cloudlets)	$N = 20$
Number of services	$K = 30$
Average data size of the services	$\frac{1}{K}\sum_k \Lambda_k = 50$
Average loads from regions	$\frac{1}{N}\sum_{i,k} n_i^k = 2{,}000$
Average computation capacity of cloudlets	$\frac{1}{N}\sum_j c_j = 1{,}800$
Average storage capacity of cloudlets	$\frac{1}{N}\sum_j u_j = 500$

We define two metrics to evaluate the algorithms: *latency* and *algorithm running time*. The latency is a normalized value defined by

$$\frac{\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{N+1} n_i^k \beta_{ij}^k \times d_{ij}}{\sum_{k=1}^{K}\sum_{i=1}^{N}(n_i^k \cdot d_{i,N+1})}, \tag{19}$$

where the numerator is the latency of the loads under the load dispatching β_{ij}^k, and the denominator is the latency of the loads if they are all served by the cloud. The normalized value is always less than one. The smaller this value is, the better the corresponding algorithm is. The algorithm running time is the execution time of the algorithm for outputting the service placement and load dispatching. This metric indicates the time complexity of the algorithms. We obtain the average latency and algorithm running time over the 2,000 test cases for each parameter setting.

Fig. 4a shows the latency of each algorithm as the average loads per region increases. The algorithms in this performance comparison contain our proposed greedy heuristic (Algorithm 1), and three baseline algorithms including the LP-based heuristic, FLP-based heuristic, and the genetic algorithms, which are described in Section 3.3. The number of iterations in the genetic algorithm is set as 4,000, which is large enough to guarantee the performance. From Fig. 4a, we can see that our proposed greedy heuristic has a better performance than others. We also observe that the latency increases as the loads from mobile uses increase. When the average loads per region are 1,000, the normalized latency is about 0.2, meaning that latency with the caching of services on the cloudlets is one fifth of the latency of requesting services directly from the cloud. The normalized latency increases to 0.4-0.5 when the loads from mobile users of each region is large. This is because the mobile users compete for the constrained resources on the cloudlets, and in such a case, a lot of loads are dispatched to the remote cloud.

Fig. 4b show that the latency increases as the size of the service set K increases. This is because when K increases, the number of replicas for each service will decrease, as the total storage of the cloudlets to place the replicas is constrained. In such a case, for each service, there exists a decreasing number of cloudlets that can serve the geo-distributed loads, and thus the latency will increase. However, Fig. 4c shows that the latency does not change much as the number of regions and cloudlets increases. This observation demonstrates the scalability of our algorithms. The performance is not influenced when the number of services scales up or scales down.

Fig. 4d shows the running time of the algorithms when the parameter N increases. We can see that our proposed greedy heuristic is the fastest one among the four algorithms. The FLP-based algorithm has less computational cost than the genetic algorithm and FP-based heuristic. We can also observe that the running time increases rapidly as the number of regions and cloudlets N increases.

5.2 Evaluation of CSPP

To realistically evaluate our online solution of CSPP, we have conduct the trace-driven simulation in this section.

5.2.1 Setup of Environment

We simulate an urban environment where the mobile users watch the videos on Youtube. We consider the videos as services. The mobile users can access the videos from the cloudlets rather than the remote Internet cloud. The service provider, e.g., Youtube here, needs to decide the placement of the videos on the cloudlets and dispatching the users' requests.

- *Datasets*. Our simulation is based on two real world datasets: the users' mobility dataset and the Youtube dataset. The mobility dataset includes the taxi data and metro data that are collected on 21 May 2014 in Shenzhen city. In specific, the taxi data contains the trajectories of the taxies at Shenzhen. The location of each taxi is reported every 5 to 15 seconds through the GPS. We choose 43,000 complete trajectories from 7:00 am to 23:00 pm. The length of the trajectories ranges from 2.5 to 78.1 km, with the average of 21.9 km. The time taken by each trajectory varies from 8 to 81 minutes, with an average of 25 minutes.

The metro data contains the information of users' journeys to commute the metro. For each journal, when and where it starts and ends are recorded. From the dataset, the total number of journeys is about 620,000 which spans from 7:00 am to 23:00 pm. The length of the journeys is about 32 minutes on average. As the metro data only contains the time and location of the start and destination of the user's journey, we need to construct the complete trajectory of the user's journey according to the train's speed and the distances between the stations along the trajectory.

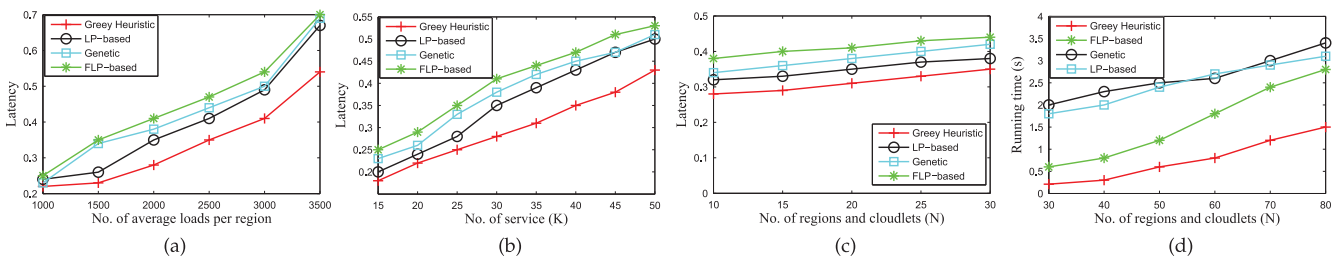


Fig. 4. Evaluation results of BSPP.

Fig. 5. The deployment of cloudlets in Shenzhen City.

The Youtube dataset [5] contains 10,324 videos that are crawled from the Youtube website on Mar 2nd, 2007. The average data size and length of the video are 13 MB and 340 seconds. The video has a recommendation list which includes its most relevant videos. We can take the Youtube dataset as a large graph, where the node represents the video and the link represents the relevance between the videos.

• *Cloudlets deployment and services configuration*. In our simulated environment, we deploy 10 cloudlets across the whole city of Shenzhen, China. Each cloudlet is located in one of the 10 administrative regions of Shenzhen, which is shown in Fig. 5. As placing one real world service often needs much more storage resources than placing one Youtube video, we do not emulate each service by one video. Instead, we partition all the videos into K sets, and emulate one service by one set. We partition the videos according to the video's *age*, which is an integer number of days between the date when the video was uploaded and Feb.15, 2007 (YouTube's establishment). The videos whose ages are located in the same period are allocated into the same set. We set K as 20 in our simulation.

• *Workload emulator*. We implement an emulator to generate the workload traces. Fig. 6 shows the process flow of the emulator. The emulator runs as a MapReduce job on the Hadoop cluster in our laboratory such that it can scale up to process larger dataset. The Map function first reads one trajectory from the mobility dataset. It then performs the *spatial formatting* for the trajectory. The fine-grained locations, e.g., the GPS coordinates in taxi data and the stations in the metro data, are transferred into the coarse-grained regions shown in Fig. 5.

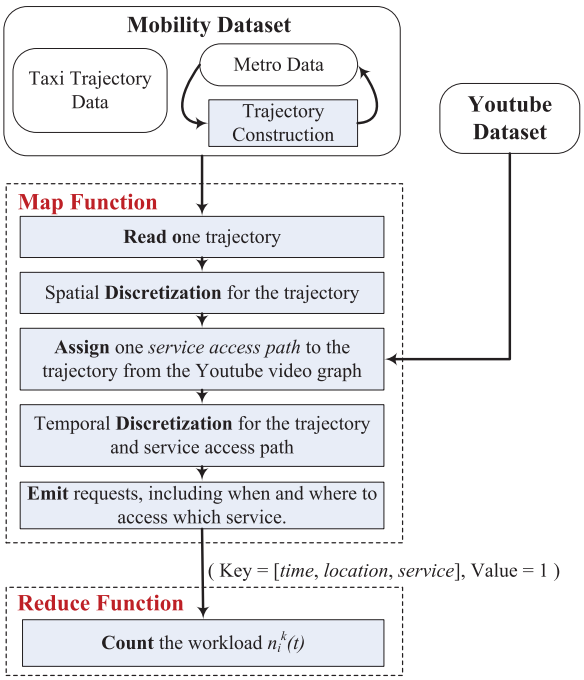


Fig. 6. The process flow to generate the workload traces.

TABLE 3
Parameters of the Simulated Environment

Parameters	Values
Number of time slots of the generated load trace	96
The duration of one time slot	10 minutes
Average load at each time slot	34,500
Variance of the load trace	8,630
Number of the services	20
Average data size of the services	6,710 MB
Average computation capacity of cloudlets	3,000
Average storage capacity of cloudlets	13,000 MB

Next, we simulate the services that the user would access when he/she moves along the trajectory. The emulator randomly assigns a *service access path* from the Youtube graph to the trajectory. We define the *service access path* as a sequence of videos and their order in which the user access the videos, which is modeled as one path in the Youtube video graph. The length of the path is the summation of the length of all the videos along the path. The assignment guarantees that the selected *service access path* has the same length with the trajectory in time domain. After the assignment, we divide the continuous time domain into discrete time slots with each being 10 mins in our simulations. The whole traces spanning from 7:00 am to 23:00 pm contains 96 time slots. In each time slot, we check the regions that the user visits and the services requested by the user when he/she visits the regions. Note that the videos on the *service access path* need to be mapped into particular services. With this information, the emulator finally emits the requests, each of which includes three properties: time, region and service. In MapReduce implementation, the Map function outputs one key-value pair for each request. The Reduce function counts the number of requests that has the same value, and outputs the workload traces $n_i^k(t)$. Table 3 shows the statistical values of the generated workload trace as well as other simulation parameters.

5.2.2 Evaluation Results

Prediction accuracy. We first evaluate the accuracy of the proposed load prediction approach. At each time slot, our method is used to predict the load distribution at the next time slot $n_i^k(t+1)$ which contains $N \times K = 200$ values in our simulation. As the load trace contains 96 time slots, we have $200 \times 95 = 19{,}000$ predictions in total. We measure the prediction accuracy by $|\widetilde{n}_i^k(t) - n_i^k(t)|$, where $\widetilde{n}_i^k(t)$ denotes the predicted value and $n_i^k(t)$ represents the true value. The red curve in Fig. 7 shows the CDF of the errors of the 19,000 predictions. Note that the average value of the true load $\sum_{i,k,t} n_i^k(t)/(N \times K \times 96)$ is 173. The curve indicates that our prediction method has the error of 24 on average, and 90 percent of the predictions have the error less than 40.

Our approach can also be used to predict the load at next Δt time slots ($\Delta t > 1$). Therefore, we also compare the

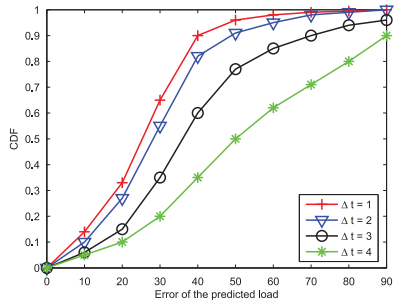


Fig. 7. CDF of load prediction error.

prediction accuracy under various Δt, i.e., $\Delta t = 2, 3, 4$, which are shown in Fig. 7. It shows that the prediction error increases obviously as the Δt increases. The result indicates that the longer period in future our method predicts, the larger error it has.

Latency and cost. We then evaluate our proposed solution for CSPP in terms of two metrics: *latency* of users' requests and *cost* of the service provider. The *latency* is normalized by Equation (19). The *cost* includes two parts: the storage resource usage on cloudlets, and the service placement transition cost over time, which are formulated in Equation (8). The former one could be the money in real world systems that is paid by service providers to the cloudlet operators/providers due to the leasing of resources on cloudlets. The latter one is the amount of data transmission incurred by the change of service placement. To simplify the metric, we measure both cost in terms of the data size which could either be the size of data stored on cloudlets or the size of data transmitted. The evaluation is based on the comparison of the four algorithms which are described as follows.

- *CSPP algorithm with predicted load (online)*. It is the proposed online algorithm for CSPP shown in Algorithm 2. At every time slot, the algorithm first predicts the load distribution at the next time slot, and then determine the optimal service placement and load dispatching. The algorithm aims to optimize both the latency and the cost of service providers.
- *CSPP algorithm with true load (offline)*. It is the offline algorithm for CSPP which takes the ground truth of load as input instead of the predictions.
- *BSPP algorithm with predicted load (online)*. It is the online algorithm for BSPP show in Algorithm 1. At each time, it first predicts the load distribution, and then takes the predicted load as input to optimize the service placement and load dispatching. The optimization objective only includes the latency, while ignoring the cost of the service provider.
- *BSPP algorithm with true load (offline)*. It is the offline algorithm for BSPP which takes the ground truth of load as input rather than the predicted load.

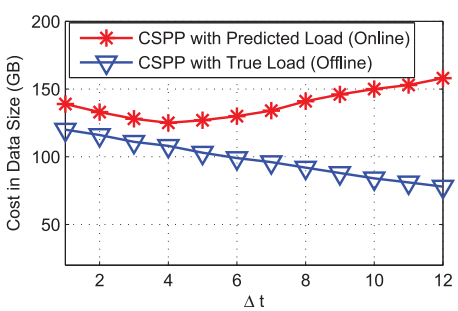


Fig. 8. Impact of Δt on cost.

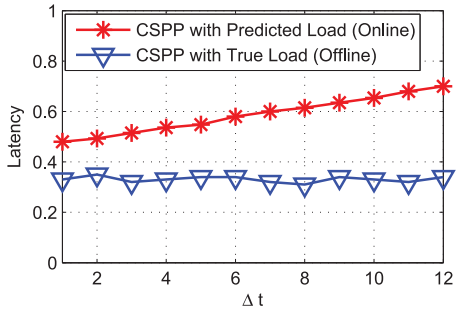


Fig. 9. Impact of Δt on latency.

Fig. 10 shows the latency of the four algorithms. The average latency of our online CSPP is about 0.5, which means that caching the service on the cloudlets can reduce the latency by 50 percent compared with directly accessing the services from the Internet cloud. We can also see that the CSPP algorithm has slightly higher latency than the BSPP algorithm under both online and offline cases. Moreover, the online CSPP algorithm has obvious longer latency than the offline algorithm due to the inevitable error in the load prediction. Fig. 10 compares the four algorithms in terms of the cost of service providers. The CSPP algorithm (Online) can reduce the cost of service provider by about 26 percent over the BSPP algorithms. If we can predict the load with more accurately, the cost could be reduced by even more, e.g., by 36 percent in the ideal case with 100 percent prediction accuracy. Through this evaluation, we conclude that taking into account of the cost by CSPP has little influence on the users' latency, but reduces the cost of service providers significantly.

Impact of Δt on the performance. Now we evaluate our online CSPP algorithm with Δt-step look ahead. Note that when $\Delta t = 1$, the online CSPP algorithm with Δt-step look ahead becomes the one-shot CSPP algorithm, which has been compared with other algorithms in the evaluations above. Fig. 9 show how Δt impacts the latency of the users' requests. We can see that as Δt increases, the performance of the online algorithm degrades. The reason is that the load prediction error can increase significantly when large Δt is applied, which leads to the increase of the latency. Unlike

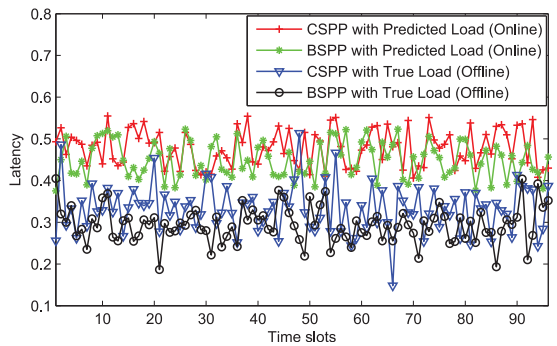


Fig. 10. The average latency of users' requests at each time slot.

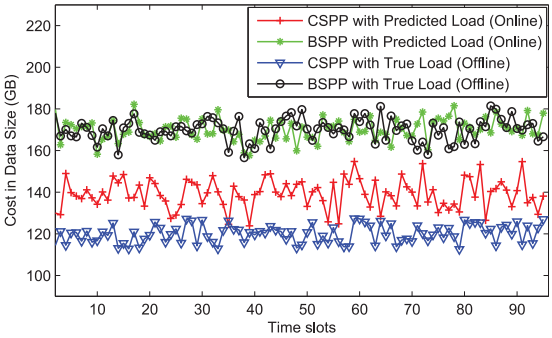


Fig. 11. The cost of service provider at each time slot.

having much influence on the performance of the online algorithm, Δt has little impact on the offline algorithm.

Fig. 8 plots the cost of service providers under different values of Δt. The cost of the online algorithm first decreases as Δt increases, and then increases when Δt is larger than 4. The influence of increasing Δt on the cost is like a double-edged sword . On one hand, increasing Δt means the consideration of longer future when determining the service placement. It brings more chances to reduce the service transition cost over the time. On the other hand, increasing Δt often causes large load prediction error, which could lead to the increase of the cost. Thus, to minimize the cost, the optimal Δt cloud neither be too great nor too little. In our simulation setup, the cost is minimized when $\Delta t = 4$. Since the latency increases as Δt increases shown in Fig. 9 , the optimal Δt is located at $(0, 4)$ when we consider both metrics of latency and cost.

6 Related Works

Cache/data placement. Cooperative caching in mobile networks are the early related works. The problem focuses on the placement of data onto the capacitated nodes and construction of corresponding pathes to access the data, such that various objectives are satisfied. Prabh and Abdelzaher [13] aim to find optimal locations to cache the data that minimize packet transmissions in wireless sensor networks. Tang et al. [14] consider the caching problem of minimizing the total data access cost in ad hoc networks. Nuggehalli et al. [15] design caching placement strategies in multi-hop wireless networks that optimally trade-off between overhead cost, i.e., energy and bandwidth, and access latency. There exist recent works studying similar problem of cache placement in cloud computing [16], [17], [18], [19]. Agarwal et al. [17] design automated mechanisms to place application data across the globally distributed data centers. Dan and Carlsson [16] consider the problem of allocating contents to the dedicated servers with limited storage and unmetered upload bandwidth. Jiao et al. [19] studies the data placement onto multi-clouds for socially aware services.

The caching problem introduced above only considers the capacity of nodes in terms of its storage or memory, i.e., being able to be placed with limited number of data items. It does not take into account the capability of nodes in terms of the amount of data access requests from other nodes they can serve. Thus, they do not deal with dispatching users' data access requests to nodes with the data replica stored. However, in our service placement, we take into account both capabilities of nodes, and jointly optimize the data placement and load dispatching among the nodes.

Load dispatching/balancing. Another related topic is on load balancing in geographically distributed data centers [10], [11], [12], [20], [21]. These works aim to allocate the workloads from different regions to the globally distributed data centers, in order to minimize particular costs of service provider, while satisfying the requirement on user-perceived latency. Qureshi [12] takes the diversity of electrical prices at various regions, and design a load dispatching system that can reduce the electric bill for Internet-Scale systems. Following this idea, many works have been done to minimize other metrics such as carbon emission, energy consumption, data transmissions and so on [10], [11], [21]. These works deal with the load dispatching under fixed data placement. In general, they assume that the distributed clouds are placed with all the required data, and the request loads are allowed to be dispatched to arbitrary cloud. However, unlike the distributed cloud, the cloudlets in the mobile cloud systems have storage capacity constraint, being able to cache limited amount of data. The load can only be assigned to the cloudlets that have cached the required data.

Content distribution in CDNs. The most related works to this paper is content distribution networks, which jointly consider the placement of contents and the assignment of users' demands to the network servers. Gao et al. [9] develop a framework that can automatically determines the placement of data objects, and also controls the fraction of user traffic directed to each datacenter. Borst et al. [23] design distributed algorithm to the problem that can approximate to the optimum of the centralized algorithm. Similar problem of placing Video-on-Demand content in CDNs have been extensively studied recently due to the increasing popularity of video web sites [8], [22]. However, these works study the joint optimization problem for the traditional datacenter workloads. In the mobile cloud system, we face with the mobile workload which is highly dynamic in both temporal and spatial domain, as the users' mobility in very fine-grained areas would cause the change of the workload distribution. This increases the complexity in the design of online algorithms that can be deployed in practical systems.

7 Conclusion

In this paper, we have studied the joint optimization of service placement and load dispatching in mobile cloud systems. We have designed an efficient heuristic algorithm to BSPP, and a set of competitive benchmark algorithms. Through extensive simulations, we demonstrate that our heuristic outperforms the benchmark algorithms in terms of access latency and algorithm run time. Based on the study of BSPP, we extend the problem to a more practical model, named CSPP, and develop an online algorithm to the problem that can be deployed easily in practical systems. We evaluate the online algorithms through the load traces that are generated based on two real world datasets, i.e., the urban transportation dataset and Youtube dataset. The results show that the online algorithm can achieve good performance in access latency and cost of service providers.

ACKNOWLEDGMENTS

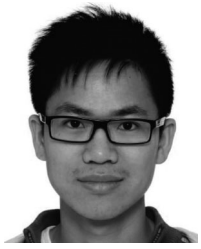
This work was partially supported by Hong Kong RGC under GRF Grant 510412, and the National High-Technology Research and Development Program (863 Program) of China under Grant 2013AA01A212.

REFERENCES

[1] L. Yang, J. Cao, S. Tang, T. Li, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM Sigmetrics Performance Evaluation Rev.*, vol. 40, no. 4, pp. 23-32, Mar. 2013.
[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.
[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. MCC Workshop Mobile Cloud Comput.*, Helsinki, Finland, Aug. 2012, pp. 13–16.
[4] Y. Ye, "An $O(n^3l)$ potential reduction algorithm for linear programming," *Math. Programming.*, vol. 50, pp. 239–258, 1991.
[5] X. Cheng, C. Dale, and J. Liu. (2007). Dataset for "Statistics and Social Network of YouTube Videos. [Online]. http://netsg.cs.sfu.ca/youtubedata/
[6] D. Shmoys, "Approximation algorithms for facility location problems," *Lecture Notes Comput. Sci.*, vol. 1913, pp. 27–32, Jun. 2003.
[7] M. Hajiaghayi, M. Mahdian, and V. Mirrokni, "The facility location problem with general cost functions," *Networks*, vol. 42, no. 1, pp. 42–47, Apr. 2003
[8] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. Lau, "Scaling social media applications into geo-distributed clouds," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 684–692.
[9] P. Gao, A. Curtis, B. Wong, and S. Keshav, "It is not easy being green," in *Proc. ACM SIGCOMM*, Aug. 2012, pp. 211–222.
[10] F. Wang, J. Liu, and M. Chen, "CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 199–207.
[11] Z. Liu, M. Lin, A. Wierman, S. Low, and L. Andrew, "Greening Geographical Load Balancing," in *Proc. ACM SIGMETRICS*, Jun. 2011, pp. 233–244.
[12] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 123–134.
[13] K. Prabh and T. Abdelzaher, "Energy-conserving data cache placement in sensor networks," *ACM Trans. Sensor Netw.*, vol. 1 no. 2, pp. 178–203, Nov. 2005.
[14] B. Tang, H. Gupta, and S. Das, "Benefit-based data caching in ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 7, no. 3, pp. 289–303, Mar. 2008.
[15] P. Nuggehalli, V. Srinivasan, C. Chiasserini, and R. Rao, "Efficient cache placement in multi-hop wireless networks," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 1045–1055, Oct. 2006.
[16] G. Dan and N. Carlsson, "Dynamic content allocation for cloud-assisted service of periodic workloads," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 853–861.
[17] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, and A. Wolman, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implementation*, Apr. 2010, pp. 17–32.
[18] A. Rasmussen, E. Kiciman, B. Livshits, and M. Musuvathi, "Improving the responsiveness of Internet services with automatic cache placement," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, Apr. 2009, pp. 27–32.
[19] L. Jiao, J. Li, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 28–36.
[20] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 854–862.
[21] M. Adnan, R. Sugihara, and R. Gupta, "Energy efficient geographical load balancing via dynamic deferral of workload," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 188–195.
[22] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. Ramakrishnan, "Optimal content placement for a large-scale VoD system," in *Proc. ACM CoNEXT*, Nov. 2010, p. 4.
[23] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[24] J. Meyer, J. Kain, and M. Wohl, *The Urban Transportation Problem*. Cambridge, MA, USA: Harvard Univ. Press, 1965.

Lei Yang received the BSc degree from Wuhan University in 2007, the MSc degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the PhD degree from the Department of Computing, Hong Kong Polytechnic University, in 2014. He is currently a postdoctoral fellow at the Department of Computing, Hong Kong Polytechnic University. His research interests include mobile cloud computing and RFID and sensor networks.

Jiannong Cao received the BSc degree from Nanjing University, China, in 1982, and the MSc and PhD degrees from Washington State University, Pullman, in 1986 and 1990, respectively, all in computer science. He is currently a chair professor and the head of the Department of Computing at Hong Kong Polytechnic University. His research interests include parallel and distributed computing, computer networks, mobile and pervasive computing, fault tolerance, and middleware. He is a fellow of the IEEE.



Guanqing Liang received the BSc degree in telecommunication engineering from Sun Yat-sen University, P.R. China, in 2011. He has been working toward the PhD degree in the Department of Computing at the Hong Kong Polytechnic University since 2012. He is currently a visiting scholar of Computer Science Department at the University of Illinois at Urban-Champaign. His research interests include pervasive computing, human behavior analysis and affective computing.

Xu Han received the BSc degree from Jilin University in 2005, and the PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2011. She is currently a postdoctoral fellow at the Department of Computing, Hong Kong Polytechnic University. Her research interests include mobile cloud computing, parallel and distributed computing, and big data analysis.