Efficient Data Cache Allocation for Sensor Clouds

Yaling Tao*, Yongbing Zhang†, Yusheng Ji‡
*† University of Tsukuba, Japan
‡ National Institute of Informatics, Japan
Email: *to.are@sk.tsukuba.ac.jp, †ybzhang@sk.tsukuba.ac.jp, ‡kei@nii.ac.jp

Abstract

A sensor cloud enables easier access to the sensor data collected from various sensor networks than in a traditional standalone sensor network. The data collected from various sensor networks are stored at the data centers of those networks but these data centers are usually distributed across the Internet. Therefore, it may take a long time for a user to access a data item if the required item is stored at a faraway data center. In this paper, we propose placing a number of data cache nodes in access networks and allocating the data needed by nearby users to minimize the data access and maintenance costs. We formulate the data allocation problem as several combinatorial optimization problems and then develop an algorithm extended from the Lagrangian relaxation method, greedy algorithm, and heuristic algorithm to solve the data allocation problems. We discuss the performance of our proposed algorithms evaluated through numerical experiments.

Index Terms

Sensor cloud, cache servers, data caching, data allocation

I. Introduction

In a traditional sensor network, the sensor data are usually stored at the base station (or the data center) of the sensor network and data access is limited to users who have direct connections to the base station. A sensor cloud combines various sensor networks based on a unified cloud computing platform and removes the limitations of such traditional sensor networks, yielding much more efficient data services [1], [2], [3], [4], [5], [6]. In a sensor cloud, the data gathered from different sensor networks are stored at data centers usually dispersed over the network, and a user can access any data no matter where he/she is. However, when a user needs a data item stored at a faraway data center, it takes a long time for him/her to obtain the required data. Therefore, it is important to have an efficient way to provide data services to users, especially when continuous or real-time data are needed in data-intensive applications for environment monitoring, healthcare, or position tracking.

Caching techniques [7], [8], [9] can be used to improve performance for data access. However, besides the data access cost for users, the cached data should be transferred from their original sources, and for real-time data, the update costs at the cache servers should not be ignored. With the collective caching approach proposed by Liao *et al.* [7], the cached data at a cache server can only be accessed by users directly connected to the server. With approaches proposed for web applications [8], [9], a user can only obtain the required data from a nearby cache server if the server holds the data; otherwise, the user has to obtain the data from its original server, yielding high access costs. Furthermore, in most previous studies, the data items are assumed to be static in nature, i.e., there is no need to update the data.

In this paper, we propose constructing a cache network by using a number of cache nodes located in access networks so that the cache network lies between the data centers of the sensor cloud and users, as shown in Fig. 1. A cache node can be an edge router with some limited storage capacity. The data gathered from a sensor network is originally stored at the data center of the network, called the *data source*, and can be stored at any cache node. We focus on the problem of how to allocate the data copies to the cache nodes in such a way that minimizes the total costs for data access and data maintenance. To this end, we formulate three optimization problems for data allocation: a single-type data allocation problem (SDAP), uncapacitated multi-type data allocation problem (UMDAP), and capacitated multi-type data allocation problem (CMDAP). We also propose an algorithm extended from the Lagrangian relaxation method, greedy algorithm, and heuristic algorithm to solve these DAPs. We evaluate the performance of these algorithms through numerical experiments, and the results show that our proposed algorithms achieve good performance with limited computation time.

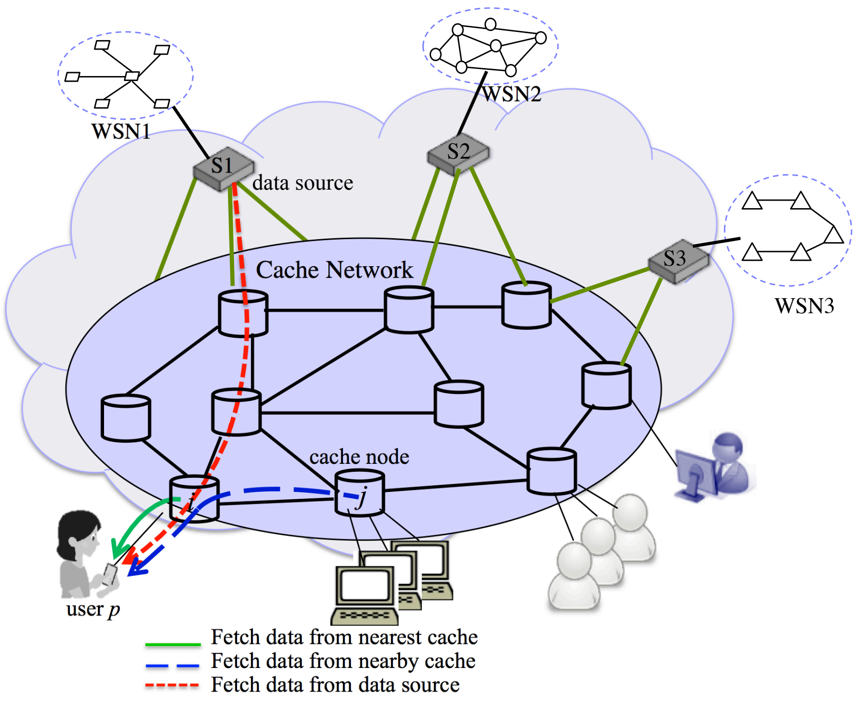

WSN1
WSN2
WSN3
S1
S2
S3
data source
Cache Network
cache node
i
j
user p
Fetch data from nearest cache
Fetch data from nearby cache
Fetch data from data source

Fig. 1. Data cache network for sensor cloud

The remainder of this paper is organized as follows. In the next section, we discuss related studies and compare them with ours. In Section III, we describe our system model and give the details of the prerequisites. The SDAP and MDAP are respectively discussed in Sections IV and V, three algorithms for these problems are described in these two sections. The numerical experiments for these problems are explained in Section VI. Finally, we conclude the paper with a brief summary in Section VII.

II. Related work

Many recent studies [1], [2], [3], [4], [5], [6] focused on how to efficiently combine sensor networks and clouds to form a sensor cloud. Hassan et al. [2] proposed a content-based publish-subscribe model in which the delivery of sensor data to users is simply implemented by a cloud-based centric broker application, while Mitton et al. [3] focused on designing a hierarchical architecture to integrate different data sources at the data centers of clouds and provide the data to users in a uniform manner. However, when a user needs a data item that is stored at a faraway data center over the network, the data access latency may become innegligibly large. Data access latency is usually a critical metric for data-intensive applications [10]; therefore it is important to have an efficient method to place the data copies at locations near the users. This paper focuses on the problem of how to allocate the copies of sensor data in access networks to minimize the total costs for data access and maintenance.

Data cache strategies have been studied in [7], [8], [9], [11], [12], [13], [14]. Bjorkqvist et al. [13] classified data items into three categories depending on the data popularity: *Gold*, *Silver*, and *Bronze*. They considered a hybrid content distribution system consisting of a central server storing all of the data items and a number of caching nodes, called *edge nodes*, each of which has a limited caching capacity. With the objective of minimizing retrieval latency, Gold data items are always stored at the edge nodes, while Bronze data items are never cached. Furthermore, Silver data items are managed locally either by a collaborative least recent used scheme or random discarding scheme. Gao et al. [14] proposed a cooperative caching scheme in which data is intentionally cached at a set of network central locations, each of which corresponds to a group of mobile nodes that can be easily accessed by other nodes in the network. These approaches take into account only the data popularity in the cache allocation decision but ignore the network topology, even though the network topology has significant impact on performance. Our proposed algorithms, on the other hand, take into account of all the costs for data access, data transmission, and data updates along with the network topology.

The DAP considered in this paper has similarities to the facility location problem (FLP) [15], [16], [17]. In the FLP, a set of facilities with facility-opening costs and a set of clients with demands are given, and the objective is to open facilities and assign clients to open facilities to minimize the facility-opening and client-assignment costs. A facility in the FLP can be considered a data item in the DAP. Furthermore, the costs for facility opening and client assignment can be represented by the costs for data placement and data access, respectively. However, there are some fundamental differences between the DAP and FLP due to the characteristics of digital data.

The data collected from a sensor network is generally updated much more often than a facility in the FLP, and the data should be transferred from the data center across the network to the cached nodes. Therefore, the data transmission cost should be taken into account in the data allocation decision. Furthermore, the computation time for data allocation is more critical than facility location decision in the FLP since the user demand may fluctuate more quickly and data lifespan may be much shorter than a facility. Additionally, the capacity of a cache node plays a key role in the DAP.

III. Data Allocation Model

The network model considered in this paper consists of a cache network along with some data sources of sensor networks as shown in Fig. 1. The data sources have all the original sensor data but may be distributed sparsely over the network. It is assumed that each sensor network generates only one type of data and the data are updated periodically. The cache network is composed of a number of cache nodes that are allocated in the access networks, i.e., near the users. Some nodes may have direct connections to a specific data source, but others have to communicate with the data source via intermediate nodes. It is assumed that a user can only access the sensor data via the nearest node to which he/she has a direct connection and can obtain the required data right away if the nearest node has the data. However, if the nearest node does not have the required data, the user has to obtain the data from another nearby node that holds the required data or in the worst case from the far away data source. To better understand how a user obtains the required data, let us see the example shown in Fig. 1. The data source s_1 stores a kind of sensor data. Each cache node, say node i or j, determines whether to cache the data depending on the users' demands around the node. A user p who has a direct connection to node i can obtain the required data if node i caches the data. If the data is not cached at node i but at node j, user p obtains the data from node j via node i. In the worst case, user p has to obtain the data from data source s_1 also via node i. It is also assumed that the cached data at each node should be updated periodically and that the storage capacity of each cache node is limited.

In this paper, three kinds of data manipulation costs are taken into account: *placement cost* for copying the data items periodically from their data sources to the cache nodes, *storage cost* for storing the data items at the cache nodes, and *access cost* for users to obtain their required data. It is assumed that the shortest path between any two nodes in the cache network is known in advance. Our objective is to determine how to allocate data copies to the cache nodes in a network to minimize the total data manipulation costs.

The sets of cache nodes and data sources are denoted by $\mathcal{N}$ and $\mathcal{S}$, respectively. Furthermore, let $\mathcal{N}' = \mathcal{S} \bigcup \mathcal{N}$. The data access and maintenance model is illustrated in Fig. 2, in which data source s_k maintains data k and node j keeps a copy of data k. The users of node i, simply denoted as node i, can obtain data k from either node j or data source s_k. The demand of node i is denoted by h_i^k. The cost for storing a data copy at node j is denoted by f_j, and the transmission costs per unit data per unit distance for data copy and data access are

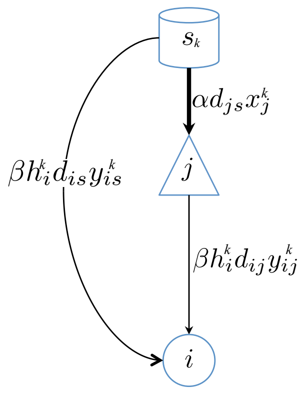


Fig. 2. Data access and maintenance model

denoted by α and β, respectively. Since the data transmission from a data source to a cache node can be done at an off-time or set with a lower priority than the data access, we assume that $\alpha \leq \beta$. The distance between nodes i and j is denoted by d_{ij}. Two variables, $x_j^k (j \in \mathcal{N}, k \in \mathcal{S})$ and $y_{ij}^k (i \in \mathcal{N}, j \in \mathcal{N}', k \in \mathcal{S})$, are used to indicate the cache allocation and data access decisions, respectively. The x_j^k value is set to 1 if data k is allocated to node j and 0 otherwise. On the other hand, the y_{ij}^k value is set to 1 if node i obtains data k from node j and 0 otherwise. The symbols used in this paper are listed in Table I.

TABLE I
SYMBOLS USED IN THIS PAPER

$\mathcal{N}$	Set of cache nodes
$\mathcal{S}$	Set of data sources
$\mathcal{N}'$	Set of data sources and cache nodes, i.e., $\mathcal{N}' = \mathcal{S} \bigcup \mathcal{N}$
h_i^k	Demand of data k at node i
Q_j	Capacity of node j
f_j	Data storage cost per unit data at node j
d_{ij}	Distance between node i and j
α	Data placement cost per unit data per unit distance
β	Data access cost per unit data per unit distance
x_j^k	Decision variable, $x_j^k = \begin{cases} 1, \text{if data } k \text{ is cached at node } j, \\ 0, \text{otherwise} \end{cases}$
y_{ij}^k	Decision variable, $y_{ij}^k = \begin{cases} 1, \text{if node } i \text{ accesses data } k \text{ from node } j, \\ 0, \text{otherwise} \end{cases}$

IV. Single-type data allocation problem (SDAP)

A. Formulation of SDAP

We first consider the SDAP in which there is only one data source, i.e., $|\mathcal{S}| = 1$. For the sake of simplicity, data source, s_k, is denoted by s, and the demand of node i, h_i^k, and the two decision variables, x_i^k and y_{ij}^k, are denoted by h_i, x_j, and y_{ij}, respectively. The SDAP can be formulated as the following integer programming problem:

$$\min \quad \alpha \sum_{j\in\mathcal{N}} d_{sj}x_j + \sum_{j\in\mathcal{N}} f_j x_j + \beta \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}'} h_i d_{ij} y_{ij}, \tag{1}$$

subject to

$$\sum_{j\in\mathcal{N}'} y_{ij} = 1, \forall i \in \mathcal{N}, \tag{2}$$

$$x_j \geq y_{ij}, \forall i, j \in \mathcal{N}, \tag{3}$$

$$x_j \in \{0,1\}, \forall j \in \mathcal{N}, \tag{4}$$

$$y_{ij} \in \{0,1\}, \forall i \in \mathcal{N}, j \in \mathcal{N}'. \tag{5}$$

Objective function (1) minimizes the total costs for data placement, data storage, and data access. Constraint (2) stipulates that node i retrieves the data from exactly one place, either the data source or a cache node. Constraint (3) means that the data request of node i cannot be served by node j unless the data is cached at node j. Furthermore, constraints (4) and (5) are the integrality constraints on the decision variables. It is well known [18], [19], [20] that the combinatorial optimization problem like problem (1) is NP-hard, and in this paper we choose to extend the Lagrangian relaxation method to obtain an approximate solution.

B. Lagrangian Relaxation Algorithm for SDAP

Lagrangian relaxation is efficient for solving difficult optimization problems and problem (1) can be reduced to an easier problem by relaxing some of its constraints, as mentioned in previous works [18], [21]. By relaxing constraint (2), i.e., by allowing a requesting node to be able to obtain a data item from more than one cache node, we have the following relaxed problem. For further details about Lagrangian relaxation, an interested reader can refer to the studies by Daskin [17] and Fisher [18], [21].

$$\max_{\lambda} \quad \min_{x,y} \quad \sum_{j\in\mathcal{N}} (\alpha d_{sj} + f_j)x_j + \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}'} (\beta h_i d_{ij} - \lambda_i) y_{ij} + \sum_{i\in\mathcal{N}} \lambda_i, \tag{6}$$

subject to

$$(3), (4), (5),$$

$$\lambda_i > 0, \forall i \in \mathcal{N}, \tag{7}$$

where $\lambda = \{\lambda_i | i \in \mathcal{N}\}$ are the Lagrange multipliers, $x = \{x_j | j \in \mathcal{N}\}$ and $y = \{y_{ij} | i \in \mathcal{N}, j \in \mathcal{N}'\}$.

For a given set of Lagrange multipliers λ, we can find the optimal solutions of problem (6). Since the optimal solutions of problem (6) are not greater than the optimal solutions of the primal problem (1) [17], [21], an optimal solution of problem (6) can be used as a *lower bound*, denoted by $LB(\lambda)$, of the primal problem. Let us first see how to determine x and y for problem (6). To minimize the value of objective function (6), if $\beta h_i d_{ij} - \lambda_i \geq 0$, we can set $y_{ij} = 0$; otherwise, $y_{ij} = 1$. Furthermore, to determine x_j, we compute $V_j = \alpha d_{sj} + f_j + \sum_i \min\{0, \beta h_i d_{ij} - \lambda_i\}$. If we set $x_j = 1$, the value of objective function (6) will be changed by V_j; therefore, if $V_j < 0$, we set $x_j = 1$; otherwise, $x_j = 0$.

The optimal solution x obtained for problem (6) can be used as a feasible solution to the primal problem (1) since it satisfies constraints (3) and (4). Furthermore, to ensure constraint (2), i.e., $\sum_{j \in \mathcal{N}'} y_{ij} = 1$, holds, we can let node i access the data at the nearest cache node or at the data source in the worst case. These x and y yield a feasible solution to the primal problem (1) and its corresponding value of objective function (1) can be used as an *upper bound*, denoted by $UB(\lambda)$, of the primal problem.

We used a subgradient optimization procedure [17] to update the Lagrange multipliers to minimize the difference between UB and LB as follows:

$$\lambda_i^{n+1} = \max\{0, \lambda_i^n - t^n(\sum_j y_{ij}^n - 1)\}, \tag{8}$$

where λ_i^{n+1} denotes the Lagrange multipliers in the nth iteration, and t^n, called a stepsize, is given by

$$t^n = \frac{\gamma^n(UB(\lambda^n) - LB(\lambda^n))}{\sum_i(\sum_j y_{ij}^n - 1)^2}, \tag{9}$$

where y_{ij}^n and γ^n are the decision variables and a constant value in the nth iteration, respectively. Furthermore, $UB(\lambda^n)$ and $LB(\lambda^n)$ are respectively the upper and lower bounds of objective function (1) in the nth iteration. If the value of $LB(\lambda^n)$ does not change after a certain number of iterations, say, a iterations, we choose to decrease the value of γ, e.g., by setting $\gamma^n = \omega\gamma^{n-a}(0 < \omega < 1)$. The initial values of λ_i and γ are given by $\lambda_i^1 = \min_{(i,j) \neq (i,i)} \beta h_i d_{ij}$ and $\gamma^1 = 2$, respectively.

Note that the Lagrangian relaxation method may not always lead to an optimal solution [21], i.e., the difference between UB and LB may not vanish. Therefore, we have to terminate the computation when the difference between UB and LB becomes less than a tolerance error ϵ_0 or a prespecified number of iterations is reached. The extended Lagrangian relaxation algorithm (LAG) for SDAP is detailed in Algorithm 1 and its computation complexity is given by $\mathcal{O}(I|\mathcal{N}|^2)$, where I denotes the maximum number of iterations.

C. Greedy Algorithm for SDAP

The proposed greedy algorithm (GRE) determines whether to place a data copy at a node depending on the costs of data placement and storage. The nodes next to the data source first trigger the algorithm and then notify their neighboring nodes of the caching decisions. The data allocation is determined according to a breadth-first order, i.e., a node closer to the data source has a higher priority to make its caching decision. When a node has made the cache decision, it is marked as a *determined node* and the set of determined nodes is denoted by N_c. The distance between an *undetermined* node i and the nodes in N_c, denoted by d_i^*, is defined by the shortest

Algorithm 1 Lagrangian relaxation algorithm for SDAP

```
1: Initialization: $n = 0, UB = UB(\lambda^n) = \infty, LB = LB(\lambda^n) = 0$
2: while $UB - LB \geq \epsilon_0$ and $n \leq I$ do
3:     $n \leftarrow n + 1$
       /* step $4 \sim 13$: calculate the $LB$ by solving the relaxed problem (6) */
4:     for each node $i \in \mathcal{N}$ do
5:         Set Lagrange multipliers $\lambda_i^n$ according to (8)
6:         for each candidate cache node $j \in \mathcal{N}$ do
7:             Calculate $V_j = \alpha d_{sj} + f_j + \sum_i \min\{0, \beta h_i d_{ij} - \lambda_i^n\}$
8:             Set $x_j^n = \begin{cases} 1, \text{if } V_j < 0, \\ 0, \text{otherwise} \end{cases}$
9:             Set $y_{ij}^n = \begin{cases} 1, \text{if } x_j^n = 1 \text{ and } \beta h_i d_{ij} - \lambda_i^n < 0, \\ 0, \text{otherwise} \end{cases}$
10:        end for
11:    end for
12:    Calculate the value of objective function (6) using $x^n$, $y^n$, $\lambda^n$, and let it to be $LB(\lambda^n)$
13:    if $LB(\lambda^n) > LB$ then $LB \leftarrow LB(\lambda^n)$
       /* step $14 \sim 19$: calculate the feasible solution of primal problem (1) and $UB$ */
14:    for each node $i \in \mathcal{N}$ do
15:        Set $x_j^{'n} = x_j^n$
16:        Set $y_{ij}^{'n} = \begin{cases} 1, \text{if } x_j^{'n} = 1 \text{ and } j \text{ is the nearest cache node of } i, \\ 0, \text{otherwise} \end{cases}$
17:    end for
18:    Calculate the value of objective function (1) using $x^{'n}$, $y^{'n}$, and let it be $UB(\lambda^n)$
19:    if $UB(\lambda^n) < UB$ then $UB \leftarrow UB(\lambda^n)$, $x_j \leftarrow x_j^{'n}$, $y_{ij} \leftarrow y_{ij}^{'n}$
20: end while
21: Let $x$ and $y$ to be the solution of primal problem (1)
```

distance from node i to the closest node in N_c, i.e., $d_i^* = \min_{j \in N_c} d_{ij}$.

Initially, only the source node s is included in N_c. All the undetermined nodes in $\mathcal{N}$ are sorted in increasing order using a breadth-first search (BFS) algorithm. For the nodes with the same hop to the data source, a node with a larger h_i has a higher priority to make the cache decision. We know that the cost for caching a data copy at node i is $f_i + \alpha d_{si}$. If node i decides to have a data copy, the nodes within the range of $\frac{1}{2} d_i^*$ from node i, denoted by N_i, will not place a data and access the data copy at node i if necessary. We treat the sum of the costs for accessing the data from node $m (m \in N_i)$ as the benefit for data placement at node i, and node i will cache the data if only the benefit is larger than the placement and storage costs, i.e.,

$$\sum_{m \in N_i} \frac{1}{2} \beta h_m d_i^* > f_i + \alpha d_{si}, \tag{10}$$

where i is also included in N_i. From relation (10), we see that node i makes the cache decision considering the surrounding nodes within a distance. As this distance is bound by $\frac{1}{2} d_i^*$, $|N_i|$ is much smaller than $|\mathcal{N}|$. The GRE is detailed in Algorithm 2, and its computation complexity is given by $\mathcal{O}(\max\{|N_i|\}|\mathcal{N}|)$.

Algorithm 2 Greedy caching algorithm for SDAP

```
1: Initialization: $N_c \leftarrow s$, sort $\mathcal{N}$ using BFS, $i = 1$
2: while $i \leq |\mathcal{N}|$ do
3:     $d_i^* = \min_{j \in N_c} d_{ij}$
4:     if $\sum_{m \in N_i} \frac{1}{2}\beta h_m d_i^* > f_i + \alpha d_{si}$ then
5:         Set $x_i = 1$
6:         Add $i$ to $N_c$
7:     else
8:         Set $x_i = 0$
9:     end if
10:    $i \leftarrow i + 1$
11: end while
```

V. Multi-type data allocation problem

It is common to have more than one type of data items, and we consider two versions of MDAPs: uncapacitated and capacitated.

A. Uncapacitated Multi-Type Data Allocation Problem (UMDAP)

When the capacity of a cache node is greater than the total amount of sensor data, we can treat the DAP as an UMDP; that is, there is no capacity limitation. Therefore, the allocation of each data item can be considered independently from one another and the data allocation problem can be formulated as follows.

$$\min \quad \sum_{k \in \mathcal{S}} (\alpha \sum_{j \in \mathcal{N}} d_{sj} x_j^k + \sum_{j \in \mathcal{N}} f_j x_j^k + \beta \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}'} d_{ij} h_i^k y_{ij}^k), \tag{11}$$

subject to

$$\sum_{j \in \mathcal{N}'} y_{ij}^k = 1, \forall i \in \mathcal{N}, k \in \mathcal{S}, \tag{12}$$

$$x_j^k \geq y_{ij}^k, \forall i, j \in \mathcal{N}, k \in \mathcal{S}, \tag{13}$$

$$x_j^k \in \{0, 1\}, \forall j \in \mathcal{N}, k \in \mathcal{S}, \tag{14}$$

$$y_{ij}^k \in \{0, 1\}, \forall i \in \mathcal{N}, j \in \mathcal{N}', k \in \mathcal{S}. \tag{15}$$

The solutions to problem (11) can be obtained using the LAG or GRE described in Section IV.

B. Capacitated Multi-type Data Allocation Problem (CMDAP)

A cache node usually has storage capacity and can only store a limited number of data items. In this paper, we assume that all the data items have the same size and can be cached at any node. The capacity of a cache node j, denoted by Q_j, indicates the number of data items node j can hold. Thus, the data allocation problem considered here can be formulated as follows.

$$\min \ \alpha \sum_{k\in\mathcal{S}} \sum_{j\in\mathcal{N}} d_{sj} x_j^k + \sum_{k\in\mathcal{S}} \sum_{j\in\mathcal{N}} f_j x_j^k + \beta \sum_{k\in\mathcal{S}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}'} d_{ij} h_i^k y_{ij}^k, \tag{16}$$

subject to

$$(12), (13), (14), (15),$$

$$\sum_{k\in\mathcal{S}} x_j^k \leq Q_j, \forall j \in \mathcal{N}. \tag{17}$$

Constraint (17) shows that the number of data items cached at node j can not exceed its storage capacity. By relaxing constraint (17) using the Lagrange multipliers μ_j in problem (16), we have the following relaxed problem.

$$\max_{\mu} \quad \min_{x,y} \sum_{j\in\mathcal{N}} \sum_{k\in\mathcal{S}} \hat{f}_j x_j^k + \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}'} \sum_{k\in\mathcal{S}} \beta h_i^k d_{ij} y_{ij}^k + \hat{Q}_j, \tag{18}$$

subject to

$$(12), (13), (14), (15),$$

$$\mu_j \geq 0, \forall j \in \mathcal{N}', \tag{19}$$

where $\mu = \{\mu_j | j \in \mathcal{N}'\}$, $x = \{x_j^k | j \in \mathcal{N}, k \in \mathcal{S}\}$ and $y = \{y_{ij}^k | i \in \mathcal{N}, j \in \mathcal{N}', k \in \mathcal{S}\}$, $\hat{f}_j = (\alpha d_{sj} + f_j - \mu_j)$ and $\hat{Q} = \sum_j \mu_j Q_j$.

We see that problem (18) can be solved similarly to the UMDAP problem if the values of μ can be fixed. Unfortunately, it is generally difficult to determine the values of μ since the solutions of different data allocations affect one another. Therefore, it is difficult to apply the Lagrangian relaxation method to this problem. In this paper, we propose a heuristic algorithm, shown in Algorithm 3, to solve the CMDAP. We first run an appropriate data allocation algorithm such as Algorithm 1 or 2 to determine the allocation for each data item $k (k \in \mathcal{S})$ without considering the capacity constraint. Accordingly, the capacity constraints of some nodes, denoted by the set of N_v, may be violated. For each node $j (j \in N_v)$, we try to reallocate some data items to its neighboring nodes to guarantee its capacity constraint. We search for the neighboring nodes of node j, denoted by the set N_j^d, that are located within a predefined distance d from node j and have spare storage spaces. Then, we choose a cache node in N_j^d, denoted by j_*^k, that yields the least cost for reallocation. This procedure is repeated until there is no capacity violation or no node with available cache. If we cannot find an appropriate node for reallocation, the data item(s) with the highest access cost will not be cached. The time complexity of the proposed heuristic algorithm is given by $\mathcal{O}(|\mathcal{S}||\mathcal{N}|)$.

VI. Numerical Experiments and Evaluations

In this section, we discuss the performance evaluation of our proposed algorithms through numerical experiments. The parameter settings used in the experiments are listed in Table II. The ratio of the costs for

Algorithm 3 Heuristic Reallocation Algorithm for CMDAP

```
1: Run an appropriate data allocation algorithm, e.g., Algorithm 1 or 2, for each data item without considering capacity constraint
2: Put the nodes that violate capacity constraints in $N_v$ and the allocated data items of node $j$ ($j \in N_v$) in a set called $S_j$
3: for each node $j \in N_v$ do
4:   while $|S_j| > Q_j$ do
5:     Search for each neighboring node $i$ that satisfies $|S_i| \leq Q_i$ and put it into $N_j^d$
6:     if $N_j^d = \emptyset$ then
7:       Delete data $k$ from node $j$, i.e., $S_j \leftarrow S_j \setminus k$, such that the value increment of objective function (16) is minimum
8:     else
9:       Find a node $j_*^k$ in $N_j^d$ to reallocate data $k$ such that the value increment of objective function (16) is minimum
10:      Reallocate data $k$ to node $j_*^k$, i.e., $S_{j_*^k} \leftarrow S_{j_*^k} + 1$, and delete $k$ from node $j$, i.e., $S_j \leftarrow S_j \setminus k$
11:    end if
12:  end while
13: end for
```

data placement, data storage and data access is determined by referring the typical service charges provided in Amazon S3 [22]. Since the data placement from a data source to a cache node can be performed at off-time period, the cost for data copy should be lower than the cost for data access. On the other hand, the cost for data storage should be much lower than that for data placement or access. In the experiments, we assumed that the data are transmitted along the shortest path between two nodes. To evaluate the performance of our proposed algorithms, we defined a metric called the *performance gap* (*PG*) to show the difference between the solution of a data allocation algorithm, denoted by *OV*, and the lower bound (*LB*) as follows:

$$PG = \frac{OV - LB}{LB}. \tag{20}$$

Note that the *LB* cannot be generally obtained; therefore, the *PG* indicates only how close the solutions of our proposed algorithms can be to the theoretical LB.

TABLE II
PRIMARY PARAMETERS USED IN EXPERIMENTS

Parameter	Value	Description
f_j	1	Data storage cost per unit data, $j \in \mathcal{N}$
α	3	Data placement cost per unit data per unit distance
β	4	Data access cost per unit data per unit distance
h_i^k	1	Demand of data k at node i, $i \in \mathcal{N}$, $k \in \mathcal{S}$
ϵ_0	1.0×10^{-5}	Tolerance error

A. Numerical Experiments for SDAP

We first examined our LAG and GRE for SDAP. A well-known integer programming solver called CPLEX [23] was used to compare with our proposed algorithms. An arbitrary network model shown in Fig.3 was used in our experiments. In this network model, there is only one data source, denoted by node 0, and nine cache nodes. The data allocation results obtained with CPLEX, LAG, and GRE are shown in Fig. 3, and the corresponding costs were 45, 45, and 48, respectively. The LB obtained with LAG was 44.76. In this experiment, both CPLEX and LAG yielded optimal solutions, and GRE obtained a feasible solution with the PG value of 7.24%. The execution times of these three algorithms were 20, 10, and 2ms, respectively.

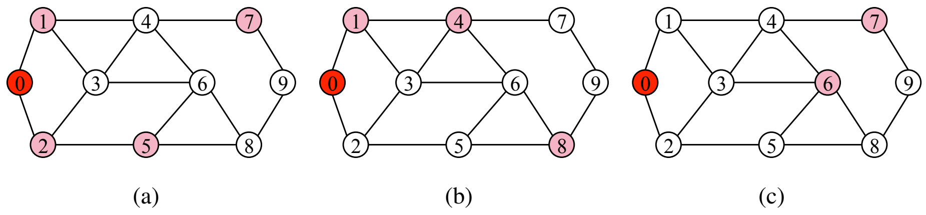
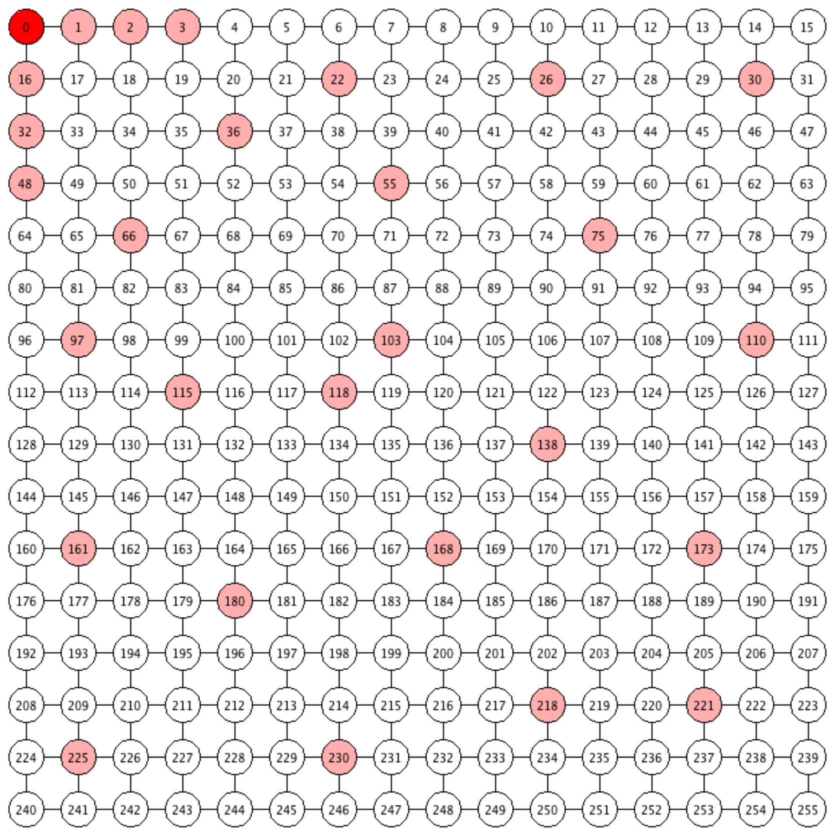
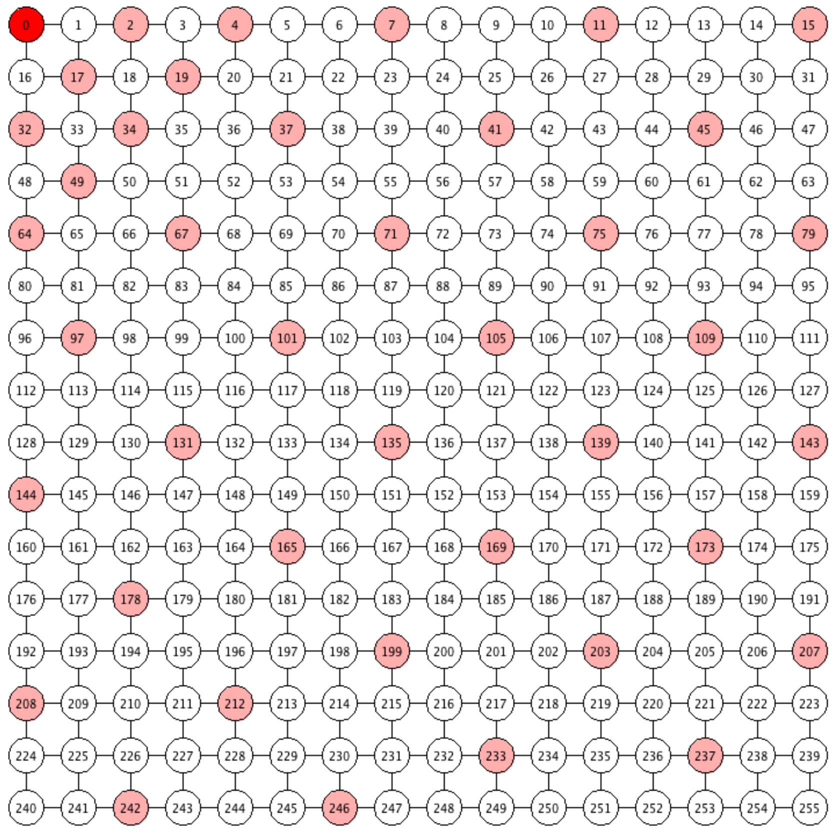


Fig. 3. Data allocation results for SADP: (a) CPLEX, (b) LAG, (c) GRE

To further investigate the performance of our proposed algorithms in a large network, we conducted experiments using a square network model of 16×16 nodes as shown in Fig. 4. Each node has at most four neighboring nodes and the data source is located at node 0. Note that all algorithms in our experiments are applicable to arbitrary topology, and the lattice network model we used is convenient for visual analysis of the experimental results. We compared the PGs and computation times of LAG and GRE with CPLEX and a random caching algorithm. With the random algorithm, a given percentage of nodes are randomly selected to place data copies, and the results shown in the figures were obtained by placing the data copies to 5, 10, and 20% of nodes (indicated as RAN5, RAN10, and RAN20 in the figures), respectively. The experimental results, including the values of objective function (16) obtained using each algorithm and its PG along with computation time, are listed in Table III, and the data allocation results obtained with LAG and GRE are displayed in Fig. 4. In this experiment, the lower bound value calculated with LAG was $2,475$. The convergence speed of the CPLEX was much slower than the others. For example, it took 170s to decrease the objective value to $2,565$, while LAG only took 22s to obtain this objective value. Since CPLEX could not obtain the optimal solution after 10-hour computation, we stopped the computation and used the best solution with a value of 2,506 obtained in the 10-hour computation period for comparison. The random algorithm with 10% data caching yielded a total cost of $3,043$ but with only a computation time of 10ms. From our experiments, we can see that LAG and GRE have distinct advantages in convergence speed compared with CPLEX and optimality compared with the random algorithm. Furthermore, the performance difference between LAG and the solution obtained with CPLEX was less than 2%.



(a)



(b)

Fig. 4. Data allocation results for SDAP in 16×16 square network: (a)LAG, (b) GRE

TABLE III
RESULTS OF VARIOUS ALGORITHMS FOR SDAP IN 16×16 SQUARE NETWORK MODEL

	Cost	Time	PG
CPLEX	2,506	>10h	1.25%
LAG	2,565	22s	3.63%
GRE	2,842	20ms	14.83%
RAN10	3,043	10ms	22.95%

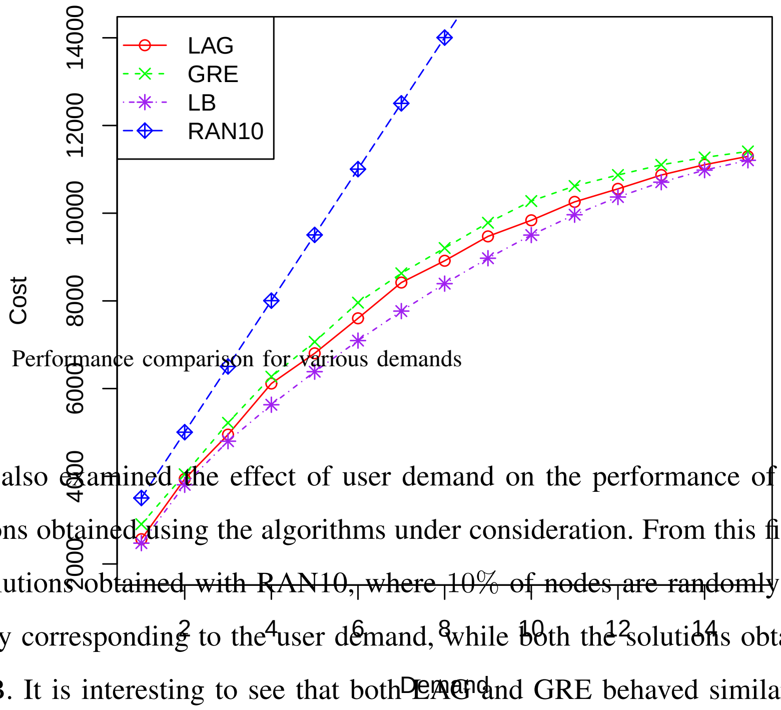


Fig. 5. Performance comparison for various demands

We also examined the effect of user demand on the performance of data allocation. Figure 5 illustrates the solutions obtained using the algorithms under consideration. From this figure, we see that the objective values of the solutions obtained with RAN10, where 10% of nodes are randomly selected to cache data items, increased linearly corresponding to the user demand, while both the solutions obtained with LAG and GRE converged to the LB. It is interesting to see that both LAG and GRE behaved similarly.

B. Numerical Experiments for MDAP

In the MDAP, we considered the same 16×16 square network model, as shown in Fig. 4, with four data items. We assumed that each node can only cache one data item, i.e., $Q_j = 1$ $(j \in \mathcal{N})$. The four data sources were placed at the corner nodes of the square network, and are indicated as red, blue, green, and yellow, respectively.

The copies of the four data items are indicated as light red, blue, green, and yellow, respectively. For UMDAP, we see that the total data allocation cost is the sum of the costs for allocating all four data items independently. Therefore, the LB of UMDAP can also be considered the sum of the LB for allocating each data item. As shown in Fig. 8, the LB is $9,900$. As mentioned earlier in Section VI, the LB is generally difficult to obtain; therefore, it is used only for comparison purposes. Due to the high computation complexity of CPLEX, we could not obtain the optimal solution with it after 24-hour computations.

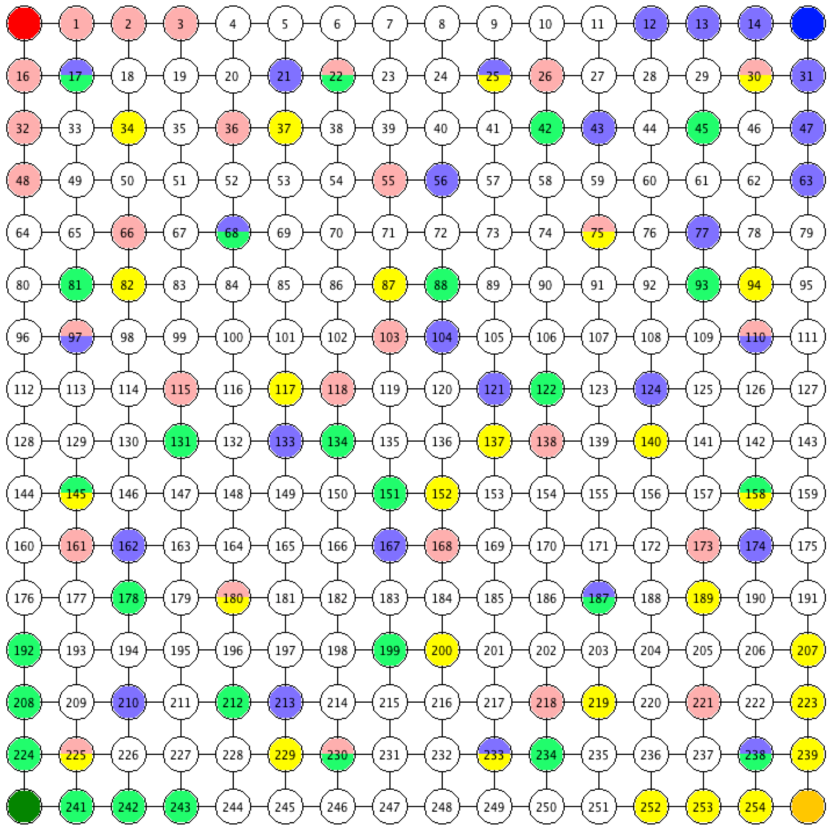
We first calculated the solutions for UMDAP, that is, we determined the data allocation by ignoring the node capacity limitation. The data allocation results for UMDAP are shown in Fig. 6. A node with more than one color means it holds more than one data copy. The solutions of UMDAP obtained with LAG and GRE are indicated by UMDAP_LAG and UMDAP_GRE, as shown in Fig. 6, and they yielded the cost values of $10,260$ and $11,368$. Note that the PGs of those two solutions were 3.64 and 14.83%, respectively.

We then executed our proposed heuristic algorithm (Algorithm 3) based on the solutions obtained for UMDAP to reallocate the data items at the nodes in which capacity constraints are violated. The solutions obtained with the heuristic algorithm are indicated as CMDAP_LAG and CMDAP_GRE, respectively, as shown in Fig. 7. The solution values obtained using the heuristic algorithm were $10,292$ and $11,228$, and the PGs of the two solutions were 3.96 and 13.41%, respectively. Note that the solutions obtained with the heuristic algorithm are comparable to those when the node capacity limitation is ignored.

TABLE IV
RESULTS OF VARIOUS ALGORITHMS FOR MDAP IN 16×16 LATTICE NETWORK MODEL

	Cost	Time	PG
UMDAP_LAG	10,260	105.43s	3.63%
CMDAP_LAG	10,292	105.66s	3.96%
UMDAP_GRE	11,368	1.13s	14.83%
CMDAP_GRE	11,228	1.60s	13.41%
RAN5	13,692	20ms	38.30%
RAN10	12,792	20ms	29.21%
RAN20	14,599	20ms	47.46%

The allocation costs obtained using the random algorithm, i.e., RAN5, RAN10, and RAN20, were $13,692$, $12,792$, and $14,599$, respectively, as shown in Table IV, and the PGs of these solutions were 38.30, 29.21, and 47.46%, respectively. The costs of MDAP are shown in Fig. 8. We see that only 10% of nodes holding data copies yielded the best performance. We see that the performance difference between CMDAP_LAG and LB was less than 4% and that CMDAP_LAG performs much better than RAN10 by 25%. Furthermore, we see that even though the computation time of CMDAP_LAG was much longer than the random algorithm but still fell within the range of reality.


(a)

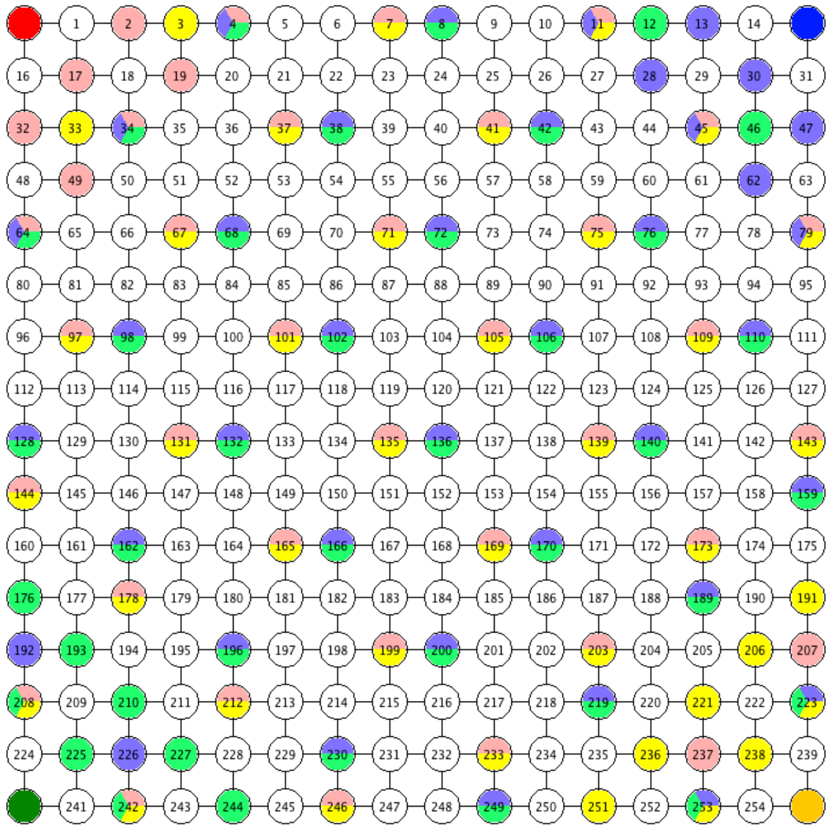
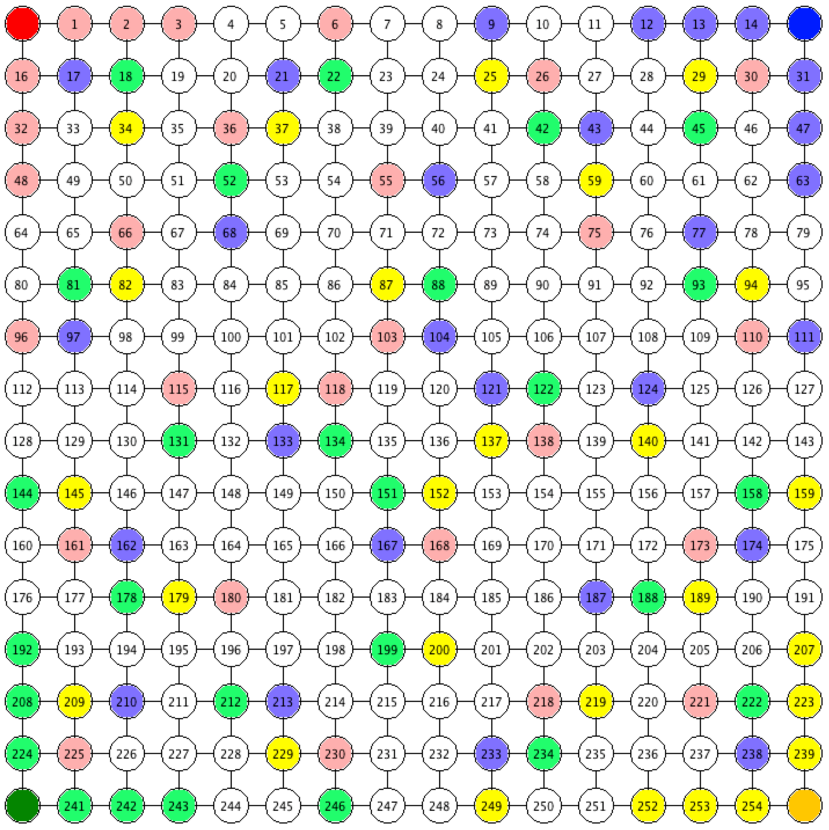

(b)

Fig. 6. Data allocation results for UMDAP in 16×16 lattice network: (a) LAG, (b) GRE


(a)

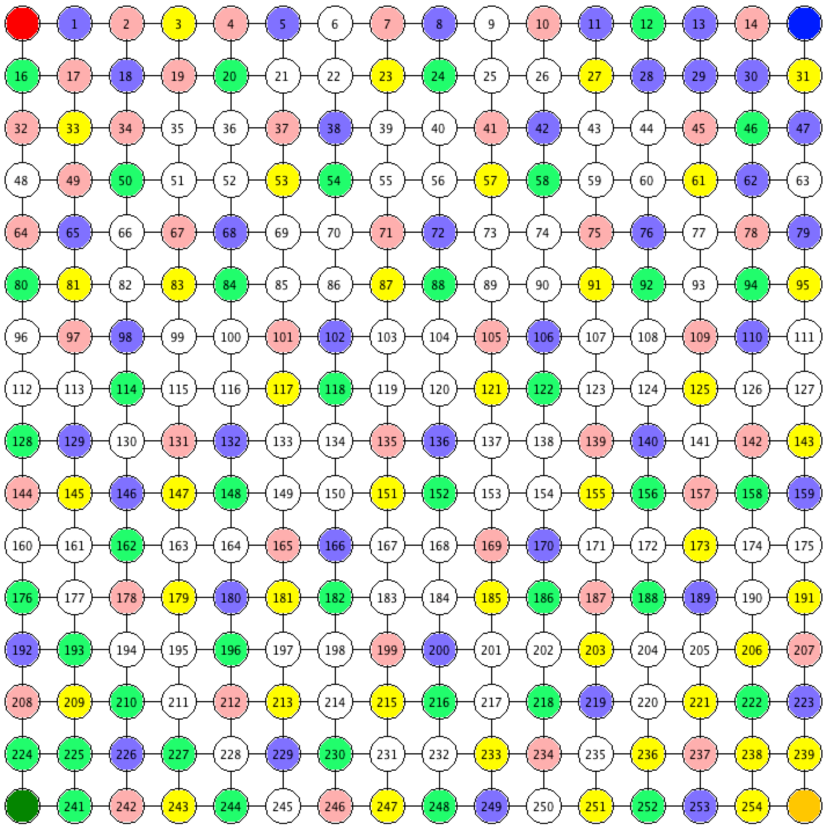

(b)

Fig. 7. Data allocation results for CMDAP in 16×16 lattice network: (a) LAG, (b) GRE

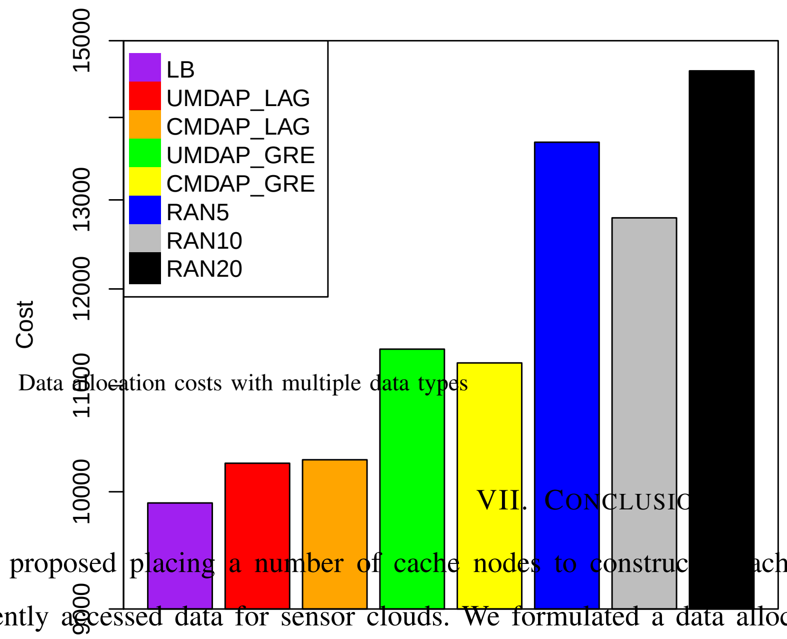


Fig. 8. Data allocation costs with multiple data types

VII. CONCLUSION

We proposed placing a number of cache nodes to construct a cache network in access networks to store frequently accessed data for sensor clouds. We formulated a data allocation problem with a single data type, SDAP, and two data allocation problems with multiple data types, UMDAP, and CMDAP. In those problems, the data placement, data storage, and data access costs are taken into account. We first extended the traditional Lagrangian relaxation algorithm and then proposed a greedy caching algorithm to solve the SDAP problem. These two algorithms were further extended to solve the UMDAP and CMDAP. We compared our proposed algorithms with a well-known integer programming solver, CPLEX, and a simple random algorithm. The experimental results show that with a single data type the performance of our extended Langrangian relaxation algorithm for SDAP is only less than 2% worse than CPLEX, but the computation time is much shorter. Furthermore, with multiple data types, the performance of our proposed heuristic algorithm for CMDAP is a little worse (less than 4%) than the LB and is similar to when the node capacity limitation is ignored. However, the computation time of our proposed heuristic algorithm for CMDAP is short enough to be accepted in real systems.

REFERENCES

[1] 'What is SensorCloud', http://sensorcloud.com/documentation, accessed September 2015

[2] Hassan, M., Song, B., Huh, E.: ’A framework of sensor-cloud integration opportunities and challenges’, Proc. Int. Conf. Ubiquitous Information Management and Communication (ICUIMC), Suwon, Korea, January 2009, pp. 618-626

[3] Mitton, N., Papavassiliou, S., Puliafito, A., Trivedi, K.: ’Combining Cloud and sensors in a smart city environment’, EURASIP Journal on Wireless Communications and Networking, 2012, 1, (1), pp. 247-256

[4] Poolsappasit, N., Kumar, V. , Madria, S., Chellappan, S.: ’Challenges in Secure Sensor-Cloud Computing’, Lecture Notes in Computer Science, 2011, 6933, pp. 70-84

[5] Kurz, M., Holzl, G., Ferscha, A.: ’Goal-Oriented Opportunistic Sensor Clouds’, Lecture Notes in Computer Science, 2012, 7566, pp. 602-619

[6] Alamri, A., Ansari, W., Hassan, M., *et al.*: ’A Survey on Sensor-Cloud: Architecture, Applications, and Approaches’, International Journal of Distributed Sensor Networks, 2013, (6), pp. 18

[7] Liao, W., Coloma, K., Choudhary, A., *et al.*: ’Collective Caching: Application-aware client-side File Caching’, Proc. Int. Symposium on High Performance Distributed Computing (HPDC), North Carolina, July 2005, pp. 81-90

[8] Sam, R., Hala, E.: ’A Quantative Study of Web Cache Replacement Strategies using Simulation’, Simulation, 2011, 88, (5), pp. 507-541

[9] Ge, C., Sun, Z., Wang, N.: ’A Survey of Power-Saving Techniques on Data Centers and Content Delivery Networks’, IEEE Commnunications Surverys and Tutorials, 2013, 15, (3), pp. 1334-1354

[10] Wang, Y., Wu, J., Tang, S., *et al.*: ’A Survey of Virtual Machine Placement in Cloud Computing for Big Data’, Ed. Yu, S., Lin, X., Misic, J., and Shen, X., (CRC Press, Taylor & Francis Group, Roca Raton, FL, 2016)

[11] Tan, B., Massoulie, L.: ’Optimal Content Placement for Peer-to-Peer Video-on-Demand Systems’, IEEE/ACM Transactions on Networking, 2011, 21, (2), pp. 566-579

[12] Lukasz, G., Marios, H., Howard, K., Barna, S.: ’Distributed data placement to minimize communication costs via graph partitioning’, Proc. Int. Conf. Scientific and Statistical Database Management (SSDBM), Aalborg, Denmark, June 2014, pp. 20

[13] Bjorkqvist, M., Chen, L., Vukolic, M., Zhang, X.: ’Minimizing retrieval latency for content cloud’, Proc. Int. Conf. Computer Communications (INFOCOM), Shanghai, China, April 2011, pp. 1080-1088

[14] Gao, W., Cao, G., Iyengar, A., Srivatsa, M.: ’Supporting Cooperative Caching in Disruption Tolerant Networks’, Proc. Int. Conf. Distributed Computing Systems (ICDCS), Minneapolis, June 2011, pp. 151-161

[15] Farahani, R., Hekmatfar, M.: ’Facility Location: Concepts, Models, Algorithms and Case Studies’ (Springer-Verlag Berlin Heidelberg 2009, New York, 1st edn., 2009)

[16] Drezner, Z., Hamacher, H.: ’Facility Location: Applications and Theory’ (Springer-Verlag Berlin Heidelberg 2004, New York, 2004)

[17] Daskin, M: ’Network and Discrete Location: Models, Algorithms, and Applications’ (Wiley-Interscience Series in Discrete Mathematics and Optimization, New York, 1995)

[18] Fisher, M.: ’The Lagrangian Relaxation Method for Solving Integer Programming Problems’, Management Science, 2004, 50, (12), pp. 1861-1871

[19] Lemarechal, C.: ’Lagrangian relaxation’, Computational combinatorial optimization, Springer Berlin Heidelberg, 2001, pp. 112-156

[20] Wolsey, L., Nemhauser, G.: ’Integer and Combinatorial Optimization’ (Wiley-Interscience, New York, NY, 1988)

[21] Fisher, M.: ’An Applications Oriented Guide to Lagrangian Relaxation’, Interfaces, 1985, 15, (2), pp. 10-21

[22] Amazon S3 Pricing, http://aws.amazon.com/s3/pricing/, accessed October 2015

[23] CPLEX Optimizer, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, accessed January 2016