2020 IEEE International Conference on Web Services (IEEE ICWS 2020), 19-23 Oct 2020, Beijing, China (c) IEEE 2020

# Budgeted Data Caching based on k-Median in Mobile Edge Computing

Xiaoyu Xia*, Feifei Chen*, Guangming Cui†, Mohamed Abdelrazek*, John Grundy‡, Hai Jin§
and Qiang He†¶(✉)
*School of Information Technology, Deakin University, Australia
†School of Software and Electrical Engineering, Swinburne University of Technology, Australia
‡Faculty of Information Technology, Monash University, Australia
§School of Computer Science and Technology, Huazhong University of Science and Technology, China
¶School of Software Engineering, Chengdu University of Information Technology, China
Email: *{xiaoyu.xia,feifei.chen,mohamed.abdelrazek}@deakin.edu.au,
†{gcui, qhe}@swin.edu.au,
‡john.grundy@monash.edu,
§hjin@hust.edu.cn

***Abstract*—In mobile edge computing (MEC), edge servers are deployed at base stations to provide highly accessible computational resources and storage capacities to nearby mobile devices. Caching data on edge servers can ensure the service quality and network latency for those mobile devices. However, an app vendor needs to ensure that the data caching cost does not exceed its data caching budget. In this paper, we present the *budgeted edge data caching* (BEDC) problem as a constrained optimization problem to maximize the overall reduction in data retrieval for all its app users within the budget, and prove that it is $\mathcal{NP}$-hard. Then, we provide an approach named IP-BEDC for solving the BEDC problem optimally based on Integer Programming. We also provide an $O(k)$-approximation algorithm, namely $\alpha$-BEDC, to find near-optimal solutions to the BEDC problems efficiently. Our proposed approaches are evaluated on a real-world data set and a synthesized data set. The results demonstrate that our approaches can solve the BEDC problem effectively and efficiently while significantly outperforming five representative approaches.**

***Keywords*-mobile edge computing, data caching, low-latency services, optimization, approximation algorithm**

## I. Introduction

The world is witnessing exponentially growing mobile traffic, which is predicted to be 32 billion connected mobile devices by 2023 [1]. This produces an enormous loads on networks, as considerable network resources are required to transmit those massive data. Two immediate consequences are increased network latency and network congestion in the mobile network. To address those issues, mobile edge computing (MEC) has emerged as a new mobile computing paradigm that allows the cloud's computational resources and storage capacities to be distributed to edge servers [2]. These edge servers, each powered by one or many physical machines, are deployed at base stations that are geographically close to mobile devices [3]. Vendors of Mobile and IoT applications (referred to as *app vendors* together hereafter) can hire computational resources and storage capacities on edge servers for hosting their apps to serve their near-by *app users* with low latency and high throughput [4]. Computation tasks from app users' mobile devices can be offloaded to nearby edge servers to ease the computation burden and energy consumption on those resource-limited devices [5]–[7].

As edge servers become most mobile devices' entry points to the Internet, a large proportion of the rapidly increasing mobile traffic data will be transmitted through those edge servers. Those edge servers provide the infrastructure for caching popular data for app users, for example, popular YouTube videos, which often accounts for a significant percentage of the mobile traffic. If the requested data is available on an edge server, a nearby app user does not have to retrieve it from the remote app server in the cloud. Caching popular data on edge servers can considerably reduce the latency in app users' data retrieval. From an app vendor's perspective, it can also largely reduce the volume of data transferred from the cloud to its app users, which may incur substantial data transfer costs [8].

Xia et al. first studied the edge data caching (EDC) problem from the app vendor's perspective. EDC aims to cache data on edge servers to cover all the app users in a specific area at minimum data caching cost [9]. A major limitation in their work is that they did not consider edge servers' storage capacities. Unlike the virtually unlimited computational resources and storage capacities in the cloud, an edge server normally owns limited computational resources and storage capacities for data caching due to its size limit [7], [10]. In the open MEC environment, many app vendors need to hire storage capacities on edge servers for caching their data. This causes fierce competition among app vendors and makes it impossible for every app vendor to cache a huge amount of data on edge servers. **In practice, an app vendor must ensure that the data caching cost does not exceed its data caching budget.** Given a budget, it is the most economic for the app vendor to cache the most popular data on edge servers because it will accommodate the most app users and produce the most data caching benefit

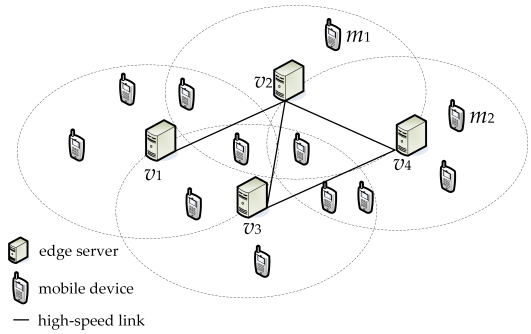



Figure 1. An example BEDC scenario

for the app vendor.

Given a piece of popular data, from the app vendor's perspective, a straightforward solution is to cache it on all the edge servers in an area for its nearby app users to retrieve. This way, their data retrieval latency can be minimized. However, this is not cost-effective, especially in an area with a lot of edge servers. In the MEC environment where nearby edge servers can communicate with each other [7], [11], an app user can retrieve data from either its nearby edge servers via zero hop or its neighbor edge servers via one hop [9] to ensure its low data retrieval latency. Given a data caching budget, the optimal data caching strategy must cache data on edge servers to minimize the app users' overall data retrieval latency without exceeding the budget. This problem is referred as the *budgeted data caching* (BEDC) problem in this paper.

Figure 1 presents an example BEDC scenario with 4 edge servers and 13 users requesting a viral video. Each edge server covers a specific geographical area. Adjacent edge servers' coverage areas partially overlap to avoid blank areas not covered by any edge servers. An user in an overlapping area can connect to one of the edge servers covering the user. This is the *server coverage constraint*. If the video required by a user is not cached on its nearby edge servers, this user can retrieve this video from their *neighbour edge servers* [9]. This is the *server adjacency constraint*. For example, the users covered by $v_1$ can also retrieve the video from the cache on $v_2$. The *data retrieval latency* is still much lower than retrieving the video from the remote cloud server. In Fig. 1, if the budget only can afford hiring one edge server to cache this video, the data should be cached on $v_2$ because $v_2$ can accommodate the most app users in this area via zero or one hop over the edge server network.

In this paper, the key contributions of this research are as follows:

- We formulate and model the BEDC problem as a constrained optimization problem from the app vendor's perspective and prove that it is $\mathcal{NP}$-hard.
- We develop an optimal approach named IP-BEDC for finding optimal solutions to BEDC problems, and an $O(k)$-approximation approach named $\alpha$-BEDC for finding approximate solutions to large-scale BEDC problems effectively and efficiently.
- We conduct extensive experiments on both a real-world data set and a synthesized data set to evaluate the proposed approaches against five representative approaches.

Table I
SUMMARY OF NOTATIONS

| Notation | Description |
|---|---|
| $\mathcal{B}$ | app vendor's data caching budget |
| $\mathcal{D}_{i,j}$ | distance from edge server $v_i$ to edge server $v_j$ |
| $\mathcal{D}_T$ | threshold of distance |
| $E$ | set of links between edge servers |
| $G$ | graph presenting a particular area |
| $K$ | total number of app users |
| $M$ | set of app users |
| $M_i$ | set of app users covered by server $v_i$ |
| $N$ | total number of edge servers |
| $S$ | set of binary variables indicating cache replicas on edge servers |
| $s_i$ | binary variable indicating cache replica on edge server $v_i$ |
| $V$ | set of edge servers |
| $v_i$ | edge server $i$ |
| $Z_m$ | maximum benefit for mobile device $m$ |
| $Z_m^i$ | benefit of caching replica on server $v_i$ for app user $m$ |
| $\mathcal{Z}(S)$ | benefit achieved by caching strategy $S$ |

The rest of this paper is organized as follows. Section II formulates the BEDC problem and proves its $\mathcal{NP}$-hardness. Section III presents and analyzes our optimal approach and approximation algorithm for solving the BEDC problem. Section IV evaluates the proposed approaches experimentally. Section V reviews the related work. Section VI concludes this paper and points out the future work.

## II. PROBLEM FORMULATION

In this section, we first formulate the BEDC problem as a constrained optimization problem (COP), then proves the $\mathcal{NP}$-hardness of this problem based on the k-median problem. The notations used in this paper are summarized in Table I.

### A. Problem Statement

In this research, the $n$ edge servers in a particular area are modeled as a graph. For each edge server $v$, the graph has a corresponding node. For each pair of linked edge servers $(v_i, v_j)$, the graph has a corresponding edge $e_{i,j}$. We use $G(V, E)$ to represent the graph, where $V$ is the set of nodes in $G$ and $E$ is the set of edges in $G$.

Similar to [9], we also model the BEDC problem generically: 1) measuring data caching cost and budget by the

number of cached data replicas; 2) measuring data retrieval latency by the number of hops on the edge server network. Take Fig. 1 as an example. The cost of caching one piece of popular data the data on all the four edge servers in Fig. 1 is 4 replicas. If the data is only cached on edge server $v_2$, mobile device $m_1$ can retrieve the data from its local edge server $v_2$ via 0 hop, while mobile device $m_2$ can retrieve the data from its neighbor edge server $v_2$ via 1 hop. This way, specific pricing models and network latency models can be easily integrated to calculate the actual data caching cost and data retrieval latency.

Given a piece of popular data, a caching strategy is presented as a vector $S =< s_1, ..., s_N >$, where $s_i \in \{0, 1\}$ denotes whether that data is to be cached on edge server $v_i$, such that $s_i = 1$ if edge server $v_i$ is selected to cache the data. As the app vendor has a finite budget for hiring storage capacities on edge servers for data caching, the total number of data replicas cached on edge servers $\forall v_i \in V$ cannot exceed the budget $\mathcal{B}$:

$$\sum_{i \in V} s_i \leq \mathcal{B} \tag{1}$$

We refer to this reduction in data retrieval latency as *data caching benefit*, which is used to evaluate the advantage of a data caching strategy. As mentioned above, the number of hops is used to measure the benefit produced by the strategy. As each edge server can only communicate with their neighbour edge servers that are directly linked to it, each app user can only retrieve cached data from an edge server within one hop, either from a nearby edge server or a neighbour edge server. Thus, the latency threshold can be defined as $\mathcal{D}_T = 2$. However, this threshold can be relaxed, i.e. $\mathcal{D}_T = 3, 4, ...$, if new techniques occur to allow data transmission through multiple edge servers rapidly while the app vendor can accept the relatively high latency. Denote $Z_m^i$ as the data caching benefit produced by caching data on $v_i$ for app user $m$ covered by edge server $v_j$ ($m \in M_j$), and it is calculated as follows:

$$Z_m^i = \max\{\mathcal{D}_T - \mathcal{D}_{i,j}, 0\} \tag{2}$$

where $\mathcal{D}_{i,j}$ is the number of hops between edge server $v_i$ and edge server $v_j$.

As discussed in Section I, an app user $m$ might be covered by multiple nearby edge servers. App user $m$ can retrieve the data from the cache on any of its nearby edge servers or their neighbour edge servers if they have the data in the cache. The data caching benefit produced by the data caching strategy for app user $m$ is the maximal number of reduced hops from $m$ to accessible edge servers that have the data in the cache:

$$Z_m = \max\{s_i \cdot Z_m^i, v_i \in V\} \tag{3}$$

Take Fig. 1 as an example. If the data is cached on $v_2$, user $m_1$ can directly access it from $v_2$ and there is $Z_{m_1} = 2$. If the data is only cached on $v_1$, $m_1$ can access it from $v_1$ via 1 hop and there is $Z_{m_1} = 1$. Moreover, if user $m_2$ cannot retrieve the data from any neighbour edge servers, it will access the remote cloud storage and there is $Z_{m_2} = 0$.

Given a data caching budget, the optimization objective, from the app vendor's perspective, is to maximize the total caching benefit $\mathcal{Z}$ of all app users produced by $S$ based on (3):

$$\text{maximize } \mathcal{Z}(S) \tag{4}$$

### *B. Problem Hardness*

Now, we prove that the BEDC problem is $\mathcal{NP}$-hard by proving the following theorem.

**Theorem 1.** *The BEDC problem is $\mathcal{NP}$-hard.*

*Proof:* To prove the BEDC problem is $\mathcal{NP}$-hard, we first introduce the classic k-median problem. The k-median problem is well-known to be $\mathcal{NP}$-hard, and can be defined as follows. Given a set of points $Set = \{1, 2, ..., N\}$, $\mathcal{A}_{i,j}$ means point $i$ is attached to point $j$ while $\mathcal{S}_i$ presents point $i$ is selected as a center. $\mathcal{P}(i, j)$ is the function for calculating the distance from point $i$ to $j$ based on the distance metric $M_D$. The formulation is displayed below:

$$object : \min \sum_{i=1}^{n} \sum_{j=1}^{n} \mathcal{A}_{i,j} \mathcal{P}(i, j) \tag{5a}$$

$$s.t. : \mathcal{A}_{i,j}, \mathcal{S}_i \in \{0, 1\}, \forall i, j = \{1, .., N\} \tag{5b}$$

$$\sum_{i=1}^{n} \mathcal{S}_i \leq k \tag{5c}$$

$$\sum_{i=1}^{n} \mathcal{A}_{i,j} = 1, \forall j = \{1, .., N\} \tag{5d}$$

$$\mathcal{A}_{i,j} \leq \mathcal{S}_i, \forall i, j = \{1, .., N\} \tag{5e}$$

Now we prove that the k-median problem can be reduced to an instance of the BEDC problem. The reduction can be done as follows: 1) adding the remote cloud server $v_{ex}$ as a node into the graph; 2) set the caching benefit obtained from $v_{ex}$ as 0. Given an instance $kMedian(Set, M_D, \mathcal{P}(i, j))$, we can construct an instance $BEDC(V, Z_K^N, benefit(v, m))$ with the reduction above in polynomial time where $|Set| = |V|$, while $Z_K^N$ is the benefit matrix that can be calculated from $M_D$ based on (2). Function $benefit(v, m)$ can be calculated from (3). This function can also project to function $\mathcal{P}(i, j)$ because the latency can be projected to the distance while the relationship between data caching benefit and latency is defined in (2). In this case, any solution $\mathcal{S}$ satisfying objective (5a) and constraint (5b), also satisfies objective (4), while constraint (5c) is equivalent to constraint (1). Moreover, app vendors who cannot access data from edge servers are attached to $v_ex$. Thus, constraints (5d) and (5e) are fulfilled.

In conclusion, any solution $\mathcal{S}$ always satisfies the reduced BEDC problem if $\mathcal{S}$ satisfies the k-median problem. Therefore, the BEDC problem is reducible from the k-median problem and it is $\mathcal{NP}$-hard. ■

## III. Strategy Formulation

We formulate our optimization model based on the BEDC problem described in the previous section. We then introduce our $O(k)$-approximate algorithm named $\alpha$-BEDC, where $k$ is a constant, and evaluate its performance theoretically.

### A. Optimization Model

For a graph $G = (V, E)$, where $V = \{v_1, .., v_N, \}$ and $E = \{e_1, ..., e_L\}$, there are a set of variables $S = \{s_1, .., s_N\}$, where $s_i = \{0, 1\}, \forall i \in \{1, ..., N\}$, $s_i$ being 1 if the a data replica is cached on the $i^{th}$ node, 0 otherwise. The constraints for the COP model are:

$$Z_m = \max(s_i * Z_m^i), \forall m \in \{1, ..., K\}, \forall i \in \{1, ..., N\} \quad (6)$$

$$0 \leq Z_m \leq \mathcal{D}_T, \forall m \in \{1, ..., K\} \quad (7)$$

$$\sum_{i=1}^{N} s_i \leq \mathcal{B} \quad (8)$$

Constraint family (6) is converted from (3). It ensures that every app user only retrieves the data from the nearest possible edge server (within $\mathcal{D}_\mathcal{T}$) hops. Constraint family (7) is calculated from (2). When an app user $m$ cannot retrieve the data from any edge server in the area, the caching benefit of this app user $b_u$ is 0. Constraint (8) guarantees that the total number of data replicas cached on the edge servers in this area must not exceed the app vendor's data caching budget.

From the app vendor's perspective, the optimization objective must produce the maximal data caching benefit:

$$\max \sum_{m=1}^{K} Z_m \quad (9)$$

The COP above can be solved with Integer Programming problem solvers, such as IBM CPLEX Optimizer.

### B. $O(k)$-approximation Algorithm

As the COP of BEDC is $\mathcal{NP}$-hard, finding the optimal solution to the COP discussed in Section III-A is intractable in large-scale BEDC scenarios. This section presents an $O(k)$-approximation algorithm, named $\alpha$-BEDC, for finding near-optimal solutions to large-scale BEDC problems effectively and efficiently, where $\alpha$ is an input parameter in the algorithm.

Algorithm 1 presents the pseudo-code of $\alpha$-BEDC. The algorithm starts with the input and initialization in Lines 1-3. Then, all possible options which first cache data on $\min\{\alpha, \mathcal{B}\}$ edge servers are collected, and the algorithm finds the candidate set with the maximal benefit (Lines 4 - 11). If the number of hired data cache spaces reaches the budget $\mathcal{B}$, this algorithm returns the results (Lines 12 - 14). Otherwise, an extra solution generated by Lines 16-21 is added to the candidate set, which selects $\beta$ edge servers based on a greedy manner presented in Algorithm 2. This way, the algorithm collects those possible solutions which have $\beta$ edge servers in Line 22. After that, the algorithm always selects the edge server which can produce the maximal data caching benefit to cache the data for each possible solution $c \in C$. This process iterates until the budget limit is reached or no benefits produced by selecting a new edge server to cache data (Lines 23-36).

In the $\alpha$-BEDC algorithm, the computational overhead of finding the $\beta$-optimal subsets in Line 11 is $C_N^{\min\{\alpha,\mathcal{B}\}}$. The iteration in Lines 21-32 will calculate at most $C_N^{\min\{\alpha,\mathcal{B}\}}$ times as there are at most $C_N^{\min\{\alpha,\mathcal{B}\}}$ subsets in $C$. For function $getServerIdWithMax\Delta Benefit()$, the algorithm always finds the edge server which can produce the most data caching benefit from all the remaining edge servers. Therefore, the computational overhead of this function is $O(KN)$. Thus, the computational complexity of this algorithm is $O(KN^{\min\{\alpha,\mathcal{B}\}+1})$.

Now, we prove that the $\alpha$-BEDC algorithm is an $O(k)$-approximation algorithm.

**Theorem 2.** *The approximation factor of the $\alpha$-BEDC algorithm is 1 if the budget is less than or equal to $\alpha$.*

*Proof:* When the budget $\mathcal{B}$ is less than or equal to $\alpha$, Algorithm 1 terminates at Line 14. In this case, $\alpha$-BEDC first collects all possible options whose size is exactly $\mathcal{B}$, then returns the possible options with the maximum benefits, which is equal to the benefits achieved by the optimal solution. Thus, the approximation factor of $\alpha$-BEDC is 1, if $\mathcal{B} \leq \alpha$. ■

Now, we only calculate the approximation ratio for $\mathcal{B} > \alpha$. Let OPT present the optimal solution of the BEDC problem, and $S_t$ denote the current solution set when $t^{th}$ edge server added by $\alpha$-BEDC.

One possible solution $S_t$ from Algorithm 1 contains edge server set $I$ (Lines 15-19). As the benefit produced by solution $S$ obtained by this algorithm is always equal to or greater than $S_t$, we get:

$$\mathcal{Z}(S) \geq \mathcal{Z}(S_t) \quad (10)$$

**Lemma 3.** *For the $t^{th}$ edge server added into $S_t$, the increase in benefit, $\Delta Z_t$, follows:*

$$\Delta Z_t \geq \frac{\mathcal{Z}(OPT) - \mathcal{Z}(S_{t-1})}{\mathcal{B}} \quad (11)$$

*Proof:* Since $S_t$ selects the edge server with the maximum benefits, for each edge server in $OPT$ but not in $S_{t-1}$, the benefit is at most $\Delta Z_t$. Since the total remaining budget is bounded by $\mathcal{B}$, the total benefit produced by the edge

**Algorithm 1** $\alpha$-BEDC

**Input** $V, M, D, \mathcal{B}, \alpha$
$S, C, Ops \leftarrow \varnothing$
$maxBenefit = 0$
**if** $\mathcal{B} \leq \alpha$ **then**
  $\beta = \mathcal{B}$
**else**
  $\beta = \alpha$
**end if**
$Ops = \{op : \forall op \in V \ \ \& \ \ |op| = \beta\}$
$maxBenefit = \max\{benefit(Ops)\}$
$C = \{op \in Ops : benefit(op) = maxBenefit\}$
**if** $\mathcal{B} \leq \alpha$ **then**
  **return** $C$
**end if**
$I \leftarrow \varnothing$;
**repeat**
  $i = getServerIdWithMax\Delta Benefit(I)$
  **if** $i \,!= -1$ **then**
    $I = I \cup v_i$
  **end if**
**until** $|I| = \beta \ || \ i = -1$
$C = C \cup I$
**for** $c \in C$ **do**
  $remaingBudget = \mathcal{B} - |c|$
  $candidate \leftarrow c$
  **repeat**
    $i = getServerIdWithMax\Delta Benefit(c)$
    **if** $i! = -1$ **then**
      $candidate = candidate \cup v_i$
    **end if**
    $remaingBudget - -$
  **until** $remaingBudget = 0 \ || \ i = -1$
  **if** $benefit(S) \leq benefit(candidate)$ **then**
    $S \leftarrow candidate$
  **end if**
**end for**
**return** $S$

servers in $OPT$ but not $S_{t-1}$ is at most $\mathcal{B}\Delta Z_t$. Thus, the above inequality holds. ■

**Lemma 4.** *The benefit achieved by $S_t$ satisfies the following inequality:*

$$\mathcal{Z}(S_t) \geq \left(1 - \left(1 - \frac{1}{\mathcal{B}}\right)^{t-1}\right) \mathcal{Z}(OPT) \tag{12}$$

*Proof:* Based on Lemma 3, the benefit achieved by $S_t$ can be calculated by:

$$\mathcal{Z}(S_t) = \mathcal{Z}(S_{t-1}) + \Delta Z_t \geq \left(1 - \frac{1}{\mathcal{B}}\right) \mathcal{Z}(S_{t-1}) + \frac{1}{\mathcal{B}} \mathcal{Z}(OPT) \tag{13}$$

**Algorithm 2** Function used in $\alpha$-BEDC

$getServerIdWithMax\Delta Benefit(c)$:
$id = -1$
**for** $v_i \in V$ **do**
  **if** $benefit(c \bigcup \{v_{id}\}) < benefit(c \bigcup \{v_i\})$ **then**
    $id = i$;
  **end if**
**end for**
**return** $id$

Thus, we can easily prove (12) by the inductive proof. The details of the proof process are omitted here. ■

If we add the $(t+1)^{th}$ edge server based on the $\alpha$-BEDC algorithm, we can obtain the following:

$$\mathcal{Z}(S_{t+1}) \geq \left(1 - \left(1 - \frac{1}{\mathcal{B}}\right)^{t}\right) \mathcal{Z}(OPT) \tag{14}$$

When $t = \mathcal{B}$, we obtain the performance of solution $S_{t+1}$:

$$\begin{aligned} \mathcal{Z}(S_{t+1}) &\geq \left(1 - \left(1 - \frac{1}{\mathcal{B}}\right)^{t}\right) \mathcal{Z}(OPT) \\ &= \left(1 - \left(1 - \frac{1}{t}\right)^{t}\right) \mathcal{Z}(OPT) \quad \geq \left(1 - \frac{1}{e}\right) \mathcal{Z}(OPT) \end{aligned} \tag{15}$$

However, the cost exceeds the budget when adding $(t+1)^{th}$ edge server into $S_{t+1}$. Now, we analyse the approximation ratio of solution $S$ provided by $\alpha$-BEDC by the following theorem.

**Theorem 5.** *If the budget $\mathcal{B}$ is greater than $\alpha$, the approximation ratio is:*

$$\frac{\alpha + 1}{\alpha}\left(1 + \frac{1}{e - 1}\right)$$

*Proof:* From (15), we obtain the benefit achieved by solution $S_t$:

$$\mathcal{Z}(S_t) \geq \left(1 - \frac{1}{e}\right) \mathcal{Z}(OPT) - \Delta Z_{t+1} \tag{16}$$

For $t = \alpha + 1, ..., \mathcal{B}$, $\Delta benefit_t$ is always no more than the benefits produced by any edge server before $s_t$. Hence,

$$\Delta Z_t \leq \frac{1}{\alpha} \mathcal{Z}(I) \leq \frac{1}{\alpha} \mathcal{Z}(S_t) \tag{17}$$

Thus, we obtain the benefit $\mathcal{Z}(S)$ produced by solution $S$ by combining (16), (17) and (10):

$$\mathcal{Z}(S) \geq \mathcal{Z}(S_t) \geq \left(1 - \frac{1}{e}\right) \mathcal{Z}(OPT) - \frac{1}{\alpha} \mathcal{Z}(S) \tag{18}$$

Therefore, we have:

$$\mathcal{Z}(S) \geq \frac{\alpha}{\alpha + 1}\left(1 - \frac{1}{e}\right) \mathcal{Z}(OPT) \tag{19}$$

Table II
PARAMETER SETTINGS

| | $n$ | $\mathcal{B}$ | $d$ | **DS** |
|---|---|---|---|---|
| **Set #1** | 10, ..., 40 | 4 | 1 | Real-World |
| **Set #2.1** | 10, ..., 40 | 4 | 1 | Synthetic |
| **Set #2.2** | 20 | 4 | 1.0, ..., 3.0 | Synthetic |
| **Set #2.3** | 20 | 2, ..., 7 | 1 | Synthetic |

and the approximation ratio is calculated as:

$$ratio = \frac{\mathcal{Z}(S)}{\mathcal{Z}(OPT)} = \frac{\alpha}{\alpha+1}\left(1-\frac{1}{e}\right) \tag{20}$$

■

**Theorem 6.** *$\alpha$-BEDC is an $O(k)$-approximation algorithm.*

*Proof:* Based on Theorem 2 and Theorem 5, the benefit achieved by the $\alpha$-BEDC algorithm is at least $\frac{\alpha}{\alpha+1}(1-\frac{1}{e})Z(OPT)$, while $\alpha$ is a constant in the input. Thus, the $\alpha$-BEDC algorithm is an $O(k)$-approximation algorithm, where $k$ is a constant. ■

Based on (20), the solution found by $\alpha$-BEDC is closer to the optimal solution with a higher $\alpha$. In Section IV, we apply $\alpha = 2$, where the approximation ratio is 42.14% in the worst case, to evaluate its effectiveness.

## IV. EXPERIMENTAL EVALUATION

We have experimentally evaluated the performance of our IP-BEDC and $\alpha$-BEDC. All experiments were conducted on a Windows-10 machine equipped with Intel Core i7-8550 processor (8 CPUs, 1.80GHz) and 8GB RAM.

### A. Experiment Settings

*1) Comparison Approaches:* In this section, we evaluate the performance of our approaches, i.e., IP-BEDC and $\alpha$-BEDC, against five representative approaches:

- *IP-BEDC*: IP-BEDC finds the optimal solution defined by Section II. It solves the COP defined in Section III-A with IBM's CPLEX Optimizer.
- *2-BEDC*: $\alpha$-BEDC finds the near-optimal solution with Algorithm 1, where $\alpha = 2$ in the experiments.
- *CDN* [12]: This approach origins from the collaborative data caching approach in content delivery network.
- *NC-BEDC*: This approach finds the optimal non-collaborative data caching solution, where app users can only access data from their nearby edge servers. Other than that, it is formulated and implemented in a similar way to IP-BEDC.
- *Greedy-Connection (GC)*: This approach always selects the edge server that has the most neighbours to cache data under the budget constraint (8).
- *Greedy-Covered-Devices (GD)*: This algorithm keeps selecting the edge server with the most app users under the budget constraint (8).
- *Random*: This algorithm keeps selecting the edge server randomly under the budget constraint (8).

*2) Experiment data:* Two sets of experiments were conducted on two data sets (**DS**), Set #1 on a real-world data set and Set #2 on a synthetic data set. The real-world data set is the widely-used EUA data set [2], [13], [14], [15]. This data set contains the geographical locations of 125 base stations and 816 mobile users in the Melbourne CBD area. The second data set is synthesized to simulate more general BEDC scenarios. In Set #2, a certain number of edge servers are randomly distributed within a particular area with app users also generated randomly. Edges are randomly generated to ensure the edge servers constitute a connected graph.

*3) Experiment parameters:* To simulate different BEDC scenarios, three parameters are varied in the experiments.

- The total number of edge servers ($n = |V|$). In experiment Set #1 and Set #2.1, this number varies from 10 to 40 in steps of 5.
- Edge density ($d = |E|/|V|$). In experiment Set #2.2, this number varies from 1 to 3 in steps of 0.4.
- Budget ($\mathcal{B}$). In experiment set 2.3 this parameter varies from 2 to 7 in steps of 1.

*4) Performance Metrics:* Three metrics are employed to evaluate all approaches, two for effectiveness and one for efficiency:

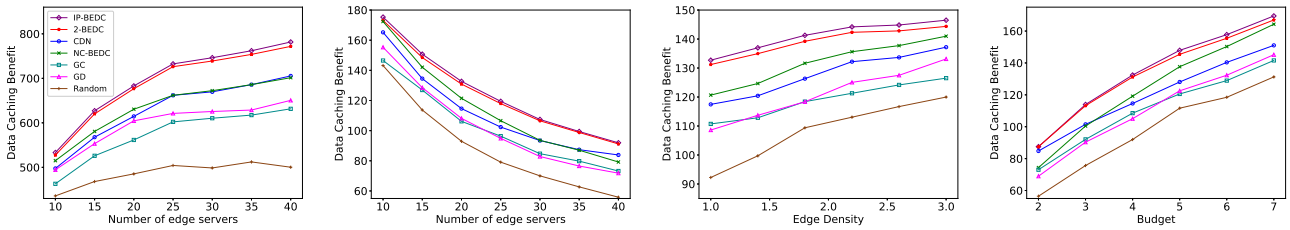
- Data caching benefit ($Z$), measured by the number of hops reduced by caching strategy, the higher the better.
- Data hit ratio ($hr$), inspired by hit ratio used in [12] [16], measured by the percentage of app users served by caching strategy, the higher the better.
- Computational overhead ($time$), measured by the time taken to find the solution, the lower the better.

Table II summarizes the parameter settings. Every time the value of a parameter varies, the experiment is repeated for 100 times and the results are averaged. According to (9), the data caching benefit is evaluated by summing the benefits of all app users. Thus, to stabilize the impact of the number of app users, we always select or generate a total of 100 app users in experiment Set #2.1, Set #2.2 and Set #2.3.

### B. Experimental Results

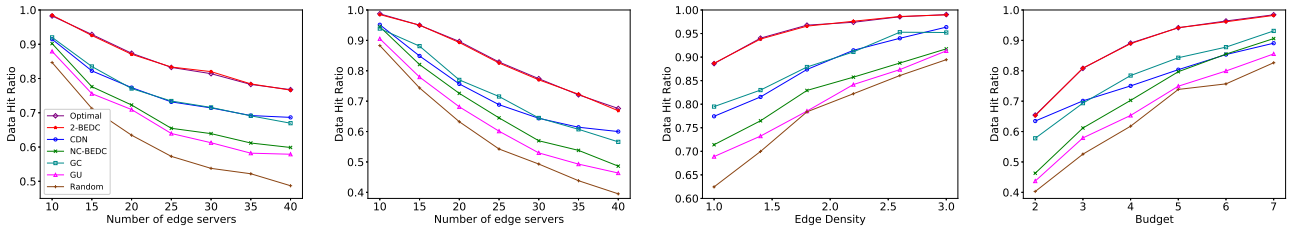
The effective results are shown in Fig. 2, Fig. 3 and the efficient results are shown in Fig. 4.

*1) Effectiveness:* The experimental results based on the real-world EUA data set are presented in Fig. 2(a), Fig. 3(a) and Fig. 4(a). Overall, **IP-BEDC achieves the highest data caching benefit and the highest data hit ratio, while the performance of 2-BEDC is very close to that of IP-BEDC.** The data caching benefits achieved by IP-BEDC and 2-BEDC in Fig. 2(a) are much higher than other five approaches. The advantages of IP-BEDC are 10.52%



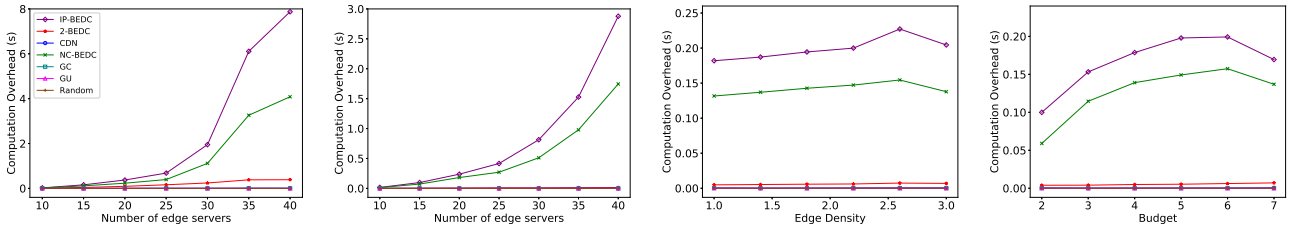

(a) Set #1 (b) Set #2.1 (c) Set #2.2 (d) Set #2.3

Figure 2. $Z$ vs. $d$




(a) Set #1 (b) Set #2.1 (c) Set #2.2 (d) Set #2.3

Figure 3. $hr$ vs. $d$




(a) Set #1 (b) Set #2.1 (c) Set #2.2 (d) Set #2.3

Figure 4. $time$ vs. $d$

against CDN, 9.40% against NC-BEDC, 21.28% against GC, 16.46% against GD and 42.93% against Random on average. 2-BEDC also outperforms those approaches significantly, by an average of 9.37%, 8.26%, 20.02%, 15.24% and 41.44% respectively. Fig3 (a) shows that the data hit ratio of 2-BEDC is almost same as that of IP-BEDC. Both IP-BEDC and 2-BEDC achieve much higher average data hit ratio than other approaches, i.e., 85.50% (IP-BEDC) and 85.45% (2-BEDC) versus 76.21% (CDN), 70.07% (NC-BEDC), 76.23% (GC), 67.96% (GD) and 61.62% (Random).

Fig. 2(b), Fig. 3(b) and Fig. 4(b) show the results of experiment Set #2.1. It shows that the data caching benefit and data hit ratio achieved by **IP-BEDC and 2-BEDC outperform all the other approaches**. When the number of edge servers increases from 10 to 40, the data caching benefit decreases for all seven approaches, from 175.32 to 91.93 by 47.56% for IP-BEDC, from 173.1 to 91.19 by 47.32% for 2-BEDC, from 165.21 to 83.87 by 49.23% for CDN, from 172.38 to 79.2 by 54.05% for NC-BEDC, from 146.52 to 73.22 by 50.03% for GC, from 155.3 to 71.8 by 53.77 % for GD and from 143.19 to 55.85 by 61.00% for Random. However, the trends of all approaches in Fig. 2(b) are different from those in Fig. 2(a). This is because the number of app users is fixed as 100 in experiment Set #2, while the number of app users in Set #1 is changed based on the real-world data set. With the increase in the number of edge servers, the average number of app users served by each edge server decreases. Thus, the data caching benefits achieved by all those approaches decrease. Fig. 3(b) demonstrates that the data hit ratio of 2-BEDC is nearly the same as that of IP-BEDC again. The advantages of IP-BEDC are 0.31% against 2-BEDC, 14.31% against CDN, 23.32% against NC-BEDC, 13.86% against GC, 30.98% against GD and 41.31% against Random.

Fig. 2(c), Fig. 3(c) and Fig. 4(c) depict the results obtained in Set #2.2 where the edge density varies. In terms of data caching benefit and data hit ratio, **IP-BEDC and 2-BEDC outperform the other approaches** with significant margins. For data caching benefit in Fig. 2(c), the average advantages of IP-BEDC are 1.39% against 2-BEDC, 10.33%

against CDN, 6.97% against NC-BEDC, 18.56% against GC, 16.54% against GD and 30.01% against Random. Fig. 2(c) and Fig. 3(c) also show that the edge density impacts all approaches in a different way from the number of edge servers. When the edge density increases from 1.0 to 3.0, the data caching benefit of IP-BEDC increases from 132.7 to 146.5 while its data hit ratio increases from 88.62% to 99.03%. The main reason for the increase is that the app users have more chance to retrieve data from an edge server via one hop. Thus, more benefits and higher hit ratio can be achieved. Moreover, the margins between our approaches and other five approaches become smaller in both data caching benefit and hit ratio as the edge density increases.

In experiment Set #2.3, **IP-BEDC achieves the highest data caching benefit and hit ratio again, followed by 2-BEDC**. When the budget increases from 2 to 7, the data caching benefit increases, as demonstrated in Fig. 2(d), from 87.47 to 169.48 for IP-BEDC, from 87.47 to 166.95 for 2-BEDC, from 84.78 to 151.03 for CDN, from 74.36 to 164.28 for NC-BEDC, from 73.04 to 141.57 for GC, from 68.97 to 145.17 for GD and from 56.43 to 131.25 for Random. In terms of data hit ratio, the advantages of IP-BEDC are 0.14% against 2-BEDC. Comparing with the performance gap between IP-BEDC and 2-BEDC, the gaps between 2-BEDC and other approaches are much larger. 2-BEDC averagely outperforms CDN by 12.99%, NC-BEDC by 20.78%, GC by 11.22%, GD by 28.53% and Random by 35.35%.

Overall, our **IP-BEDC and 2-BEDC significantly and consistently outperform CDN, NC-BEDC, GC, GD and Random** in formulating cost-effective data caching strategies in different BEDC scenarios. As an approximate approach, the effectiveness of 2-BEDC is about 98.63% to 99.98% of IP-BEDC in all the experiments. This is acceptable in most, if not all, cases, especially in large-scale BEDC scenarios where finding the optimal solutions is impractical for IP-BEDC.

*2) Efficiency:* The efficiency is evaluated by the average computational overhead when finding a solution to the BEDC problem. The results are presented in Fig. 4. As demonstrated by Fig. 4(a) and Fig. 4(b), **IP-BEDC is much more computationally expensive than all the other approaches**. This validates the $\mathcal{NP}$-hardness of the BEDC problem - excessive computational overheads are inevitable for finding the optimal solution to large-scale BEDC problems. In BEDC scenarios with the most number of edge servers, IP-BEDC takes 7.87 seconds to find the optimal solution in Set #1, as shown by Fig. 4(a), and 2.88 seconds in Set #2.1, as shown by Fig. 4(b). Interestingly, the computation overheads of both IP-BEDC and NC-BEDC decrease when the edge density exceeds 2.5 in Fig. 4(c). The reason is that it is more likely to cover all edge servers via one hop. This way, the corresponding computation time decreases. Similar in Fig. 4(d), the computational overheads of both IP-BEDC and NC-BEDC increase when the budget increases from 2 to 6, then decrease after that, as illustrated in Fig. 4(d). The reason is that a budget of 6 allows IP-BEDC to find a solution to accommodate all the app users. The extra budget makes it easier for IP-BEDC to find any one of the multiple optimal solutions. **In comparison with IP-BEDC, 2-BEDC is much more efficient**, with an average of at most 0.39 seconds to find the near-optimal solutions in all the scenarios.

## V. Related Work

As an extension of cloud computing, the computing resources and services are distributed by Mobile Edge Computing (MEC) [17]. The problem of computation offloading arises, which has been well studied with consideration of edge servers' energy efficiency, offloading cost and joint service with caching [5] [18] [19].

Recently, the challenges raised by data caching are investigated in the MEC environment. Existing approaches from conventional networking environments and cloud computing cannot be directly implemented in the MEC environment with new characteristics. Thus, researchers have proposed and investigated new ideas and techniques of data caching. In [20], the authors proposed their approaches for ensuring the quality of time-sensitive multimedia transmissions over the 5G wireless network by integrating in-network caches and edge caches. Zhang et al. [21] proposed a cooperative edge caching architecture to improve the function of edge caches by utilizing computation resources. A caching scheme was introduced by the authors as well to provide edge caching services with implementation of the smart vehicles. Zhang et al. [20] integrated in-network caching and edge caching to ensure the latency requirements of time-sensitive transmissions over the 5G network. Drolia et al. [22] proposed a caching system, namely Cachier, to minimize the data retrieval latency. They implemented a coordinating mechanism to balance the loads between the cloud server and edge servers dynamically. The authors of [21] introduced a new edge caching architecture with improved resource utility by using smart vehicles as external edge caches.

Edge computing inherits the pay-as-you-go pricing model from cloud computing. Thus, the cost incurred for app vendors is critical to the success of edge computing because they are the main users of the edge servers. However, all the above work tackle the data caching problem from the mobile network operator's or the app user's perspective. We made the first attempt to tackle the data caching problem from the app vendor's perspective in the edge computing environment with the aim to cover all the app users in an area [9], [23]. However, due to app vendors' competition for the limited computational resources on edge servers, it is unrealistic to always accommodate all the app users' requests regardless of the data caching cost or budget. In

this work, we solves the new BEDC problem from the app vendor's perspective to maximum its data caching benefit, while fulfilling the budget constraint, the server coverage constraint and the server adjacency constraint.

## VI. Conclusion

In this paper, we formulated the new Budgeted Edge Data Caching (BEDC) problem in the mobile edge computing environment as a constrained optimization problem from the app vendor's perspective. We proved that the BEDC problem is $\mathcal{NP}$-hard. To solve this problem, we proposed an optimal approach named IP-BEDC based on the Integer Programming technique to maximum the data caching benefit measured by the overall reduction in app users' data retrieval latency. As the BEDC problem is $\mathcal{NP}$-hard, we also provided an approximate approach named $\alpha$-BEDC for finding sub-optimal solutions to large-scale BEDC problems more efficiently. Extensive experiments were conducted on a widely-used real-world data set and a synthetic data set to evaluate the performance of the proposed approaches. The results showed that our approaches significantly outperformed five representative approaches in various BEDC scenarios.

This research has established the foundation for the BEDC problem and opened up a number of future research directions. In our future work, we will consider the mobility of mobile devices, dynamic data popularity, real-time cache updating scenarios, security constraints and data regulation.

## Acknowledgement

This research is partially funded by Australian Research Council Discovery Projects No. DP170101932, DP180100212 and Laureate Fellowship FL190100035.

## References

[1] P. Cerwall, P. Jonsson, R. Möller, S. Bävertoft, S. Carson, I. Godor, P. Kersch, A. Kälvemark, G. Lemne, and P. Lindberg, "Ericsson mobility report," *On the Pulse of the Networked Society. Hg. v. Ericsson*, 2015.

[2] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *International Conference on Service-Oriented Computing*, 2018, pp. 230–245.

[3] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[4] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, 2017.

[5] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2018, pp. 207–215.

[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[7] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[8] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," *Nature reviews genetics*, vol. 11, no. 9, pp. 647–657, 2010.

[9] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based optimal data caching in edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 477–493.

[10] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[11] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4514–4526, 2018.

[12] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2016.

[13] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 86–101.

[14] P. Lai, Q. He, G. Cui, F. Chen, M. Abdelrazek, J. Grundy, J. Hosking, and Y. Yang, "Quality of experience-aware user allocation in edge computing systems: A potential game," in *40th IEEE International Conference on Distributed Computing Systems*. IEEE, 2020.

[15] G. Cui, Q. He, X. Xia, F. Chen, H. Jin, and Y. Yang, "Robustness-oriented k edge server placement," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2020.

[16] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," in *13th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, 2017, pp. 165–172.

[17] M. Yannuzzi, F. van Lingen, A. Jain, O. L. Parellada, M. M. Flores, D. Carrera, J. L. Pérez, D. Montero, P. Chacin, A. Corsaro *et al.*, "A new era for cities with fog computing," *IEEE Internet Computing*, vol. 21, no. 2, pp. 54–67, 2017.

[18] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2018.

[19] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[20] X. Zhang and Q. Zhu, "Hierarchical caching for statistical qos guaranteed multimedia transmissions over 5g edge computing mobile wireless networks," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 12–20, 2018.

[21] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5g networks with mobile edge computing," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 80–87, 2018.

[22] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 276–286.

[23] Y. Liu, Q. He, D. Zheng, M. Zhang, F. Chen, and B. Zhang, "Data caching optimization in the edge computing environment," in *26th IEEE International Conference on Web Services*. IEEE, 2019, pp. 99–106.