Tight Approximability of the Server Allocation Problem for Real-Time Applications

Ito, Takehiro
Tohoku University

Kakimura, Naonori
Keio University

Kamiyama, Naoyuki
Kyushu University

Kobayashi, Yusuke
University of Tsukuba

他

出版情報：Algorithmic Aspects of Cloud Computing (Lecture Notes in Computer Science). LNCS 10739, pp.41-55, 2018-02-28. Springer
バージョン：
権利関係：

Tight Approximability of the Server Allocation Problem for Real-Time Applications

Takehiro Ito[1⋆], Naonori Kakimura[2⋆⋆], Naoyuki Kamiyama[3⋆⋆⋆], Yusuke Kobayashi[4†], Yoshio Okamoto[5‡], and Taichi Shiitada[5]

[1] Tohoku University, Japan. takehiro@ecei.tohoku.ac.jp
[2] Keio University, Japan. kakimura@math.keio.ac.jp
[3] Kyushu University, Japan. kamiyama@imi.kyushu-u.ac.jp
[4] University of Tsukuba, Japan. kobayashi@sk.tsukuba.ac.jp
[5] University of Electro-Communications, Japan.
okamotoy@uec.ac.jp, shiitada@gmail.com

Abstract. The server allocation problem is a facility location problem for a distributed processing scheme on a real-time network. In this problem, we are given a set of users and a set of servers. Then, we consider the following communication process between users and servers. First a user sends his/her request to the nearest server. After receiving all the requests from users, the servers share the requests. A user will then receive the data processed from the nearest server. The goal of this problem is to choose a subset of servers so that the total delay of the above process is minimized. In this paper, we prove the following approximability and inapproximability results. We first show that the server allocation problem has no polynomial-time approximation algorithm unless $\mathbf{P} = \mathbf{NP}$. However, assuming that the delays satisfy the triangle inequality, we design a polynomial-time $\frac{3}{2}$-approximation algorithm. When we assume the triangle inequality only among servers, we propose a polynomial-time 2-approximation algorithm. Both of the algorithms are tight in the sense that we cannot obtain better polynomial-time approximation algorithms unless $\mathbf{P} = \mathbf{NP}$. Furthermore, we evaluate the practical performance of our algorithms through computational experiments, and show that our algorithms scale better and produce comparable solutions than the previously proposed method based on integer linear programming.

⋆ Supported by JST CREST Grant Number JPMJCR1402, Japan, and JSPS KAKENHI Grant Number JP16K00004.
⋆⋆ Supported by JST ERATO Grant Number JPMJER1201, Japan, and by JSPS KAKENHI Grant Number JP17K00028.
⋆⋆⋆ Supported by JST PRESTO Grant Number JPMJPR14E1, Japan.
† Supported by JST ERATO Grant Number JPMJER1201, Japan, and by JSPS KAKENHI Grant Numbers JP16K16010 and JP16H03118.
‡ Supported by Kayamori Foundation of Informational Science Advancement, JST CREST Grant Number JPMJCR1402, Japan, and JSPS KAKENHI Grant Numbers JP24106005, JP24700008, JP24220003, JP15K00009.

1 Introduction

Nowadays, various kinds of services such as video conferencing and document sharing are offered through a network, which are getting indispensable in our daily life. They provide an online platform for users to exchange their information with each other. Some of online services, such as online games and ticket reservation systems, are accessed by many users at the same time, which requires a real-time process.

An online service is usually maintained on one central application server to keep its integrity and maintainability. It brings in a heavy computing load on the server. Not only that, more crucially, it will cause different latencies on users. Since it takes more time for a user further from the server to send her request, a request of a further user arrives later even when she sent the request earlier. Since the ordering of requests is essential in real-time processing such as online games, we have to wait for a request from the farthest user to arrive before processing any request. After receiving all the requests from users, the server can process them in the ordering of requested time, which is estimated from the arrival times. Thus latencies of a server response occur due to the order adjustment of requests, which makes a real-time interaction inefficient.

Recently, Kawabata, Chatterjee, and Oki [11] proposed a distributed processing scheme for a real-time network to reduce the latency. In this scheme, instead of having one central server, we set up a set of application servers that provide the same service. It allows every user to access to one of the servers quickly. However, since each user sends their request to different servers, each server needs to multi-cast the received requests to the other servers to synchronize them. It enables each server to process all the requests in the right order of the requested times. A user will then receive the data processed successfully from the nearest server. Thus, the proposed scheme has an additional step to synchronize the data among the servers, but, since each user has a close server, we can reduce the latency caused by the order adjustment.

Let us define the problem more formally. We are given a set U of users and a set S of possible places that we can allocate our application servers. For each pair $i, j \in S \cup U$, we denote by $d(i, j)$ the delay of the communication between i and j. We assume that $d(i, j) \geq 0$, $d(i, j) = d(j, i)$, and $d(i, i) = 0$ for any i, j. Suppose that we choose a set X of servers in S. Then each user u accesses to her nearest server in X, which is denoted by $\rho(u, X)$. That is, $\rho(u, X) \in \operatorname{argmin}\{d(u, s) \mid s \in X\}$. As we have to wait for the last request, it takes $\max_{u \in U} d(u, \rho(u, X))$ time to receive all the requests. After that, the servers in X communicate with each other for synchronization, which takes $\max_{s,t \in X} d(s, t)$ time. Finally, the processed data is returned to every user in $\max_{u \in U} d(u, \rho(u, X))$ time. Thus the total delay of the whole process, denoted by $\mathsf{delay}(X)$, is defined to be

$$\mathsf{delay}(X) := \max_{u \in U} 2 \cdot d(u, \rho(u, X)) + \max_{s,t \in X} d(s, t).$$

We want to find a set X of servers that minimizes the total delay. The problem is called *the server allocation problem for real-time applications*, but *the server allocation problem* for short.

Kawabata, Chatterjee, and Oki [11] introduced the server allocation problem and formulated it as an integer linear programming problem. They also performed numerical simulations to measure the performance of their distributed server allocation scheme. Furthermore, Ba, Kawabata, Chatterjee, and Oki [2] later proved that, when each application server has a limited capacity of users, the problem is **NP**-hard by reduction from 3SAT. However, it was not known whether the server allocation problem is polynomial-time solvable or not.

In this paper, we show that the server allocation problem is **NP**-hard. In fact, it is impossible to get any polynomial-time approximation algorithm unless $\mathbf{P} = \mathbf{NP}$. On the positive side, assuming that the delay d is metric, we design a polynomial-time constant-factor approximation algorithm for the problem. For a subset Z in $S \cup U$, we say that the delay d is *metric in* Z if for every triple of elements i_1, i_2, i_3 in Z, d obeys the *triangle inequality*, i.e., $d(i_1, i_2) + d(i_2, i_3) \geq d(i_1, i_3)$. We consider the following three cases (see also Table 1).

Case 1: Metric in $S \cup U$. We design a polynomial-time $\frac{3}{2}$-approximation algorithm when d is metric in $S \cup U$. Moreover, we prove that, for any constant $r < \frac{3}{2}$, there exists no polynomial-time r-approximation algorithm unless $\mathbf{P} = \mathbf{NP}$.

Case 2: Metric in S. We design a polynomial-time 2-approximation algorithm when d is metric in S. Moreover, we show that, for any constant $r < 2$, there exists no polynomial-time r-approximation algorithm unless $\mathbf{P} = \mathbf{NP}$.

Case 3: General Case. We show that there exists no polynomial-time r-approximation algorithm for any r unless $\mathbf{P} = \mathbf{NP}$.

The inapproximability can be strengthened if we assume the *exponential-time hypothesis*, which basically states that we cannot solve 3SAT in $2^{o(n)}$ time [9, 10]. Namely, we show that those inapproximability results follow even if we relax the running times to $2^{o(n)}$, where n is the number of servers, assuming the exponential-time hypothesis holds.

Table 1. Summary of Our Result

	Condition	Approximability	Inapproximability
Case 1	Metric in $S \cup U$	1.5	$1.5 - \varepsilon$
Case 2	Metric in S	2	$2 - \varepsilon$
Case 3	General	—	∞

Let us conclude this section with describing related work. The server allocation problem was proposed by Kawabata, Chatterjee, and Oki [11]. They introduced a more general problem, in which users may give up sending a request if they are far from the chosen servers. Our problem is a special case of their problem when all users have to access to one of the chosen servers.

Our problem is related to the *k-center problem*. In the k-center problem, we are given a set of users and a set of facilities. The goal is to find a set X

of at most k facilities that minimizes the maximum distance from the users to X. The k-center problem is **NP**-hard, but has polynomial-time 2-approximation algorithms [7, 8] if the set of users and facilities satisfies the triangle inequality. Furthermore, for any constant $r < 2$, there exists no polynomial-time r-approximation algorithm unless $\mathbf{P} = \mathbf{NP}$ [7, 8]. Our problem differs in that we have no constraints on the number of opened facilities. Instead, the more we open facilities, the more communication cost between facilities is required, which is taken into the objective value. The *k-median problem* is also related to our problem. In this problem, we are given a set of users and a set of facilities. Then, the goal of this problem is to find a set X of at most k facilities that minimizes the sum of distances from the users to X. If the set of users and facilities satisfies the triangle inequality, then the current best polynomial-time approximation ratio for this problem is $2.611 + \varepsilon$ due to Byrka *et al.* [3]. Our problem differs in that our goal is to minimize the maximum delay.

The *hub location problem* is the problem of locating hubs and allocate non-hub nodes to decrease the transportation cost between a pair of nodes. The problem has been studied in transportation and telecommunication systems (see e.g., [1, 6] for recent surveys). The *k-hub center problem* is a variant of the hub location problem in which we locate k hubs so that the maximum distance between any pair of nodes is minimized [4, 12]. Recently, Chen *et al.* [5] proposed polynomial-time constant-factor approximation algorithms for the k-hub center problem.

2 Approximation Algorithms

In this section, we design approximation algorithms for the server allocation problem.

Let X^* be an optimal solution. We denote

$$k^* = \max_{u \in U} d(u, \rho(u, X^*)) \quad \text{and} \quad \ell^* = \max_{s,t \in X^*} d(s, t).$$

Then, $\mathsf{delay}(X^*) = 2k^* + \ell^*$.

Also, define $D_1 := \{d(s, u) \mid (s, u) \in S \times U\}$ and $D_2 := \{d(s, t) \mid (s, t) \in S^2\}$. It should be noted that $|D_1| = O(|S||U|)$, $|D_2| = O(|S|^2)$, $k^* \in D_1$ and $\ell^* \in D_2$.

2.1 Case 1: Metric in $S \cup U$

In this section, we assume that the delay d is metric in $S \cup U$, that is, for every triple of elements i_1, i_2, i_3 in $S \cup U$, the triangle inequality holds, i.e., $d(i_1, i_2) + d(i_2, i_3) \geq d(i_1, i_3)$. The idea of our algorithm is to perform two different algorithms, and then the better one of the two results gives a $\frac{3}{2}$-approximate solution.

The first algorithm simply finds a server s that minimizes $\mathsf{delay}(\{s\})$.

Lemma 1. *Let $Z_1 := \{s'\}$, where $s' \in \operatorname{argmin}\{\mathsf{delay}(\{s\}) \mid s \in S\}$. Then, $\mathsf{delay}(Z_1) \leq 2k^* + 2\ell^*$.*

Proof. Let s be a server in the optimal solution X^*. Then, by the definition of Z_1, we have

$$\mathsf{delay}(Z_1) \leq \mathsf{delay}(\{s\}) = \max_{u \in U} 2 \cdot d(s,u).$$

Since d is metric in $S \cup U$, it follows that

$$d(s,u) = d(u,s) \leq d(u, \rho(u, X^*)) + d(\rho(u, X^*), s) \leq k^* + \ell^*,$$

where the last inequality follows from the fact that $d(u, \rho(u, X^*)) \leq k^*$ and $d(\rho(u, X^*), s) \leq \ell^*$ as $\rho(u, X^*) \in X^*$. Therefore, combining the above two inequalities, we obtain $\mathsf{delay}(Z_1) \leq 2k^* + 2\ell^*$. □

For the second algorithm, we first consider

$$X_u := \{s \in S \mid d(s,u) \leq k^*,\ d(s,u') \leq k^* + \ell^*\ (\forall u' \in U)\},$$

for each user u in U, and define $Z_2^* := \bigcup_{u \in U} X_u$. Intuitively, for each user u in U, the set X_u is a set of servers that are close to u and the other users simultaneously.

Lemma 2. *Suppose that Z_2^* is defined as above. Then, $\mathsf{delay}(Z_2^*) \leq 4k^* + \ell^*$.*

Proof. Let s be a server in Z_2^*. By definition of Z_2^*, there exists a user u_s such that $d(s, u_s) \leq k^*$. Moreover, for any other server t in Z_2^*, it holds that $d(u_s, t) \leq k^* + \ell^*$. Since d is metric in $S \cup U$, we have

$$d(s,t) \leq d(s,u_s) + d(u_s,t) \leq k^* + (k^* + \ell^*) = 2k^* + \ell^*.$$

Hence it holds that $\max_{s,t \in Z_2^*} d(s,t) \leq 2k^* + \ell^*$.

Let u be a user in U. Define $s := \rho(u, X^*)$. Then, $d(u,s) \leq k^*$ holds by definition. Since d is metric in $S \cup U$, for every user u' in U, $d(u',s) \leq d(u', \rho(u', X^*)) + d(\rho(u', X^*), s)$. By definition of k^*, we have $d(u', \rho(u', X^*)) \leq k^*$. Moreover, since $\rho(u', X^*) \in X^*$, we see $d(s, \rho(u', X^*)) \leq \ell^*$. Therefore, we obtain $d(u',s) \leq k^* + \ell^*$. This, together with that $d(u,s) \leq k^*$, implies $s \in X_u$, and thus $s \in Z_2^*$. Hence, since $d(u, \rho(u, Z_2^*)) \leq d(u,s) \leq k^*$, we obtain $\max_{u \in U} 2d(u, \rho(u, Z_2^*)) \leq 2k^*$.

Therefore, it follows that

$$\mathsf{delay}(Z_2^*) = \max_{u \in U} 2 \cdot d(u, \rho(u, Z_2^*)) + \max_{s,t \in Z_2^*} d(s,t) \leq 2k^* + (2k^* + \ell^*) = 4k^* + \ell^*. \ \square$$

To find the set Z_2^*, we need to know k^* and ℓ^* in advance. We search for the exact values of k^* and ℓ^* by enumerating all the elements in D_1 and D_2, respectively. For each candidate, we construct the corresponding set Z_2^*. Since $k^* \in D_1$ and $\ell^* \in D_2$, at least one candidate returns the set X with $\mathsf{delay}(X) \leq 4k^* + \ell^*$.

Our algorithm for Case 1 is described as Algorithm 1. We now show the approximation factor of $\frac{3}{2}$.

Theorem 1. *Let Z be the output of Algorithm 1. Then, $\mathsf{delay}(Z) \leq \frac{3}{2} \cdot \mathsf{delay}(X^*)$.*

Algorithm 1 Algorithm for Case 1 (Metric in $S \cup U$)

Step 1. Compute D_1 and D_2.
Step 2. Find $Z_1 := \{s'\}$, where $s' \in \operatorname{argmin}\{\mathsf{delay}(\{s\}) \mid s \in S\}$.
Step 3. For each element $k \in D_1$ and each element $\ell \in D_2$, define

$$X_u^{k,\ell} := \{s \in S \mid d(s,u) \le k,\ d(s,u') \le k + \ell\ (\forall u' \in U)\} \text{ for each user } u \text{ in } U,$$

and construct $X^{k,\ell} := \bigcup_{u \in U} X_u^{k,\ell}$.
Step 4. Find $\min\{\mathsf{delay}(X^{k,\ell}) \mid (k,\ell) \in D_1 \times D_2\}$, and let Z_2 be the set $X^{k,\ell}$ attaining the minimum.
Step 5. If $\mathsf{delay}(Z_1) < \mathsf{delay}(Z_2)$, then return $Z := Z_1$, and otherwise, return $Z := Z_2$.

Proof. Since $\mathsf{delay}(Z_2) \le \mathsf{delay}(Z_2^*)$, Lemmas 1 and 2 imply that

$$\begin{aligned}\mathsf{delay}(Z) &\le \min\{\mathsf{delay}(Z_1), \mathsf{delay}(Z_2^*)\} \le \frac{1}{2}(\mathsf{delay}(Z_1) + \mathsf{delay}(Z_2^*)) \\ &\le \frac{1}{2}((2k^* + 2\ell^*) + (4k^* + \ell^*)) = \frac{3}{2}(2k^* + \ell^*) = \frac{3}{2} \cdot \mathsf{delay}(X^*). \qquad \square\end{aligned}$$

Since $|D_1|$ and $|D_2|$ are polynomially bounded by $|S|$ and $|U|$, Algorithm 1 runs in polynomial time. More specifically, Step 1 takes $O(|S|^2 + |S||U|)$ time. In Step 2, $\mathsf{delay}(\{s\})$ can be computed in $O(|U|)$ time for each $s \in S$, and thus Step 2 takes $O(|S||U|)$ time. In Step 3, $X_u^{k,\ell}$ can be found in $O(|S||U|)$ time for each $k \in D_1$, $\ell \in D_2$, and $u \in U$, and thus Step 3 takes $O(|S||U|) \times O(|D_1||D_2||U|) = O(|S|^4|U|^3)$ time. In Step 4, $\mathsf{delay}(X^{k,\ell})$ can be computed in $O(|S|^2 + |S||U|)$ time for each $(k,\ell) \in D_1 \times D_2$, and thus Step 4 takes $O(|S|^2 + |S||U|) \times O(|D_1||D_2|) = O(|S|^5|U| + |S|^4|U|^2)$ time. Step 5 takes $O(1)$ time as we already know $\mathsf{delay}(Z_1)$ and $\mathsf{delay}(Z_2)$. Hence, in total, Algorithm 1 runs in $O(|S|^5|U| + |S|^4|U|^3)$ time. In Section 4.1, we improve to $O(|S|^2(|S| + |U|))$.

2.2 Case 2: Metric in S

In this section, we assume that the delay d is metric in S (not necessarily in $S \cup U$). Similarly to Case 1, we use the values k^* and ℓ^* of the optimal solution X^*.

Theorem 2. *Define $X_s := \{t \in S \mid d(s,t) \le \ell^*\}$ for each server s in S, and define $Z_3^* := X_{s'}$ where $s' \in \operatorname{argmin}\{\mathsf{delay}(X_s) \mid s \in S\}$. Then, it holds that $\mathsf{delay}(Z_3^*) \le 2k^* + 2\ell^* \le 2 \cdot \mathsf{delay}(X^*)$.*

Proof. Let s be a server in the optimal solution X^*. Since $\mathsf{delay}(Z_3^*) \le \mathsf{delay}(X_s)$, it suffices to show $\mathsf{delay}(X_s) \le 2k^* + 2\ell^*$.

Since $s \in X^*$, we have $d(s,t) \le \ell^*$ for every server t in X^*. This implies that $X^* \subseteq X_s$. Hence, for every user u in U, $d(u, \rho(u, X_s)) \le d(u, \rho(u, X^*)) \le k^*$. Furthermore, since d is metric in S, for every pair of servers t, t' in X_s,

$$d(t,t') \le d(t,s) + d(s,t') \le \ell^* + \ell^* = 2\ell^*,$$

Algorithm 2 Algorithm for Case 2 (Metric in S)

Step 1. Compute D_2.

Step 2. For each element $\ell \in D_2$, define

$$X_s^\ell := \{t \in S \mid d(s,t) \le \ell\}$$

for each server $s \in S$. Find $\min\{\mathsf{delay}(X_s^\ell) \mid s \in S\}$, and let X^ℓ be the set attaining the minimum.

Step 3. Find $\min\{\mathsf{delay}(X^\ell) \mid \ell \in D_2\}$, and return the set attaining the minimum as Z_3.

where the last inequality follows since $s, t, t' \in X_s$. Therefore,

$$\mathsf{delay}(X_s) = \max_{u \in U} 2 \cdot d(u, \rho(u, X_s)) + \max_{t,t' \in X_s} d(t,t') \le 2k^* + 2\ell^*. \qquad \square$$

To find the set Z_3^*, we enumerate all the elements in D_2 to search for the exact value of ℓ^*. Our algorithm for Case 2 is described as Algorithm 2. It follows from Theorem 2 that the output of Algorithm 2 is a 2-approximate solution. Moreover, the algorithm runs in polynomial time, as $|D_2| = O(|S|^2)$. More specifically, Step 1 takes $O(|S|^2)$ time. In Step 2, the determination of X_s^ℓ takes $O(|S|)$ time and the computation of $\mathsf{delay}(X_s^\ell)$ takes $O(|S||U|)$ time for each $\ell \in D_2$ and $s \in S$, and thus Step 2 takes $O(|D_2||S|^2|U|) = O(|S|^4|U|)$ time. Step 3 takes $O(|D_2|) = O(|S|^2)$ time. Hence, the running time of Algorithm 2 is bounded by $O(|S|^4|U|)$.

In Section 4.1, we improve the running time to $\tilde{O}(|S|(|S| + |U|))$, where the $\tilde{O}$-notation suppresses the polylogarithmic factor. Note that $\Omega(|S|(|S| + |U|))$ time is needed to read off the entire delay d.

3 Hardness of approximation

In this section, we prove the inapproximability of the server allocation problem by reduction from the well-known **NP**-complete problem 3SAT.

Assume that we are given an instance I of 3SAT with variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$, where $m = O(n)$. From the instance, we construct an instance of the server allocation problem I' as follows. We define $S := \{x_1, x_2, \ldots, x_n, \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$ and $U := \{C_1, C_2, \ldots, C_m\}$. For each pair of servers s, s' in S, we define

$$d(s,s') := \begin{cases} \alpha & \text{if } \{s,s'\} \neq \{x_i, \overline{x_i}\} \text{ for any } i, \\ \beta & \text{otherwise,} \end{cases}$$

where $\alpha < \beta$. Furthermore, for each server s in S and each user u in U, we define

$$d(s,u) := \begin{cases} \gamma & \text{if } s \text{ is a literal in } u, \\ \delta & \text{otherwise,} \end{cases}$$

where $\gamma < \delta$. Then the following lemma holds.

Lemma 3. *A* 3SAT *instance I is a yes-instance if and only if the optimal value of the instance I' is at most $2\gamma + \alpha$. On the other hand, I is a no-instance if and only if the optimal value of I' is at least* $\min\{2\gamma + \beta, 2\delta + \alpha\}$.

Proof. Assume that I is a yes-instance. Then, there exists a satisfying assignment $a\colon \{x_1, x_2, \ldots, x_n\} \to \{\text{true}, \text{false}\}$ for I. We define $X := \{x_i \mid a(x_i) = \text{true}\} \cup \{\overline{x_i} \mid a(x_i) = \text{false}\}$. Since the set X has either x_i or $\overline{x_i}$ for every i, it holds that $\max_{s,s' \in X} d(s, s') \leq \alpha$. Moreover, by definition of d, we have $\max_{u \in U} d(u, \rho(u, X)) \leq \gamma$. Thus the optimal value of I' is at most $2\gamma + \alpha$. Conversely, if the optimal value of I' is at most $2\gamma + \alpha$, then there exists a set X of servers such that $\max_{s,s' \in X} d(s, s') \leq \alpha$ and $\max_{u \in U} d(u, \rho(u, X)) \leq \gamma$. Hence, X contains either x_i or $\overline{x_i}$ for every i. We set x_i to be true if $x_i \in X$ and false otherwise. This is a satisfying assignment since each clause C_j has a server within the distance γ.

The second statement follows since the objective value of I' can only be one of the following four values: $2\gamma + \alpha$, $2\delta + \alpha$, $2\gamma + \beta$, and $2\delta + \beta$. □

The above lemma immediately implies the **NP**-hardness for the server allocation problem. Moreover, the reduction introduces the gap of at least $\min\{2\gamma + \beta, 2\delta + \alpha\}/(2\gamma + \alpha)$. By setting the appropriate values for those four parameters, we obtain the tight inapproximability results.

Theorem 3. (1) *For any r, there is no polynomial-time r-approximation algorithm for the server allocation problem unless* $\mathbf{P} = \mathbf{NP}$.

(2) *Suppose that d is metric in $S \cup U$. Then, for any constant $r < \frac{3}{2}$, there exists no polynomial-time r-approximation algorithm for the server allocation problem unless* $\mathbf{P} = \mathbf{NP}$.

(3) *Suppose that d is metric in S. Then, for any constant $r < 2$, there exists no polynomial-time r-approximation algorithm for the server allocation problem unless* $\mathbf{P} = \mathbf{NP}$.

Proof. (1) Set $\alpha := 1$, $\beta := \infty$, $\gamma := 1$, and $\delta := \infty$ in the definition of d. It follows from Lemma 3 that we cannot distinguish $2\gamma + \alpha$ and $\min\{2\gamma + \beta, 2\delta + \alpha\}$ in polynomial time unless $\mathbf{P} = \mathbf{NP}$. The ratio of the two values is

$$\frac{\min\{2\gamma + \beta, 2\delta + \alpha\}}{2\gamma + \alpha} = \frac{\min\{\infty, \infty\}}{3} = \infty.$$

Thus we cannot have a polynomial-time r-approximation algorithm for any r.

(2) Set $\alpha := 2$, $\beta := 4$, $\gamma := 1$, and $\delta := 3$ in the definition of d. Note that d is metric in $S \cup U$, as we may assume that a 3SAT instance I does not contain a clause having both x_i and $\overline{x_i}$ for some i. It follows from Lemma 3 that we cannot distinguish $2\gamma + \alpha$ and $\min\{2\gamma + \beta, 2\delta + \alpha\}$ in polynomial time unless $\mathbf{P} = \mathbf{NP}$. The ratio of the two values is

$$\frac{\min\{2\gamma + \beta, 2\delta + d\}}{2\gamma + \alpha} = \frac{\min\{6, 8\}}{4} = \frac{3}{2}.$$

(3) Set $\alpha := 1$, $\beta := 2$, $\gamma := \varepsilon$, and $\delta = \infty$ in the definition of d, where $\varepsilon > 0$ is an arbitrarily small constant. Then, d is metric in S. It follows from Lemma 3 that, unless $\mathbf{P} = \mathbf{NP}$, we cannot have a polynomial-time r-approximation algorithm, where r is a constant less than

$$\frac{\min\{2\gamma + \beta, 2\delta + \alpha\}}{2\gamma + \alpha} = \frac{\min\{2\varepsilon + 2, \infty\}}{2\varepsilon + 1} \to 2 \qquad (\text{as } \varepsilon \to 0). \qquad \square$$

In our reduction in Lemma 3, the number of servers is $2n$ and the number of users can be $m = O(n)$. Thus, we may conclude that if the server allocation problem can be solved in $2^{o(n)}$ time, where n is the number of servers, then the exponential-time hypothesis fails (see [9, 10] for the precise definition). The same consequence can be obtained for the inapproximability, which we summarize in the following theorem.

Theorem 4. *If any of the following* (1)–(3) *holds, then the exponential-time hypothesis fails. Here, n is the number of servers.*
(1) *The server allocation problem can be approximated by any r in $2^{o(n)}$ time.*
(2) *The server allocation problem can be approximated by any constant $r < 2$ in $2^{o(n)}$ time when d is metric in S.*
(3) *The server allocation problem can be approximated by any constant $r < 3/2$ in $2^{o(n)}$ time when d is metric in $S \cup U$.* $\square$

Note that the existence of an exact algorithm running in $O(2^{|S|}\text{poly}(|S|, |U|))$ time is clear: We just need to go through all the possible candidates $X \subseteq S$, compute their delay, and take the minimum.

4 Experiments

Since our algorithms are combinatorial and simple, we conducted computational experiments to evaluate the practical performance of our algorithms. The performance is measured in terms of execution time and the objective value of an obtained solution. The baseline is an integer linear programming (ILP) formulation given by Kawabata et al. [11]. The ILP instances were solved by IBM ILOG CPLEX 12.7.0.0. The implementation was done by C++ (gcc 5.4.0), and all the programs were run on a machine with the following specification: OS Ubuntu 16.04.1 LTS 64 bit, Memory 3.8 GiB, Processor Intel Core i3-2120 CPU 3.30 GHz × 4, HDD 488.0 GB.

4.1 Acceleration of the Proposed Algorithms

A naive implementation of our algorithms is quite slow. As described before, Algorithm 1 for Case 1 runs in $O(|S|^5|U| + |S|^4|U|^3)$ time, and Algorithm 2 for Case 2 runs in $O(|S|^4|U|)$ time. To speed-up the algorithms without losing approximability, we inserted preprocessing steps to sort the sets D_1 and D_2.

Algorithm 3 Accelerated Algorithm 1

Step 1-1. Compute D_1 and D_2 and sort D_1 and D_2 in descending order.

Step 1-2. For each server s in S, find

$$u_s^{\min} \in \operatorname{argmin}\{d(s,u) \mid u \in U\} \quad \text{and} \quad u_s^{\max} \in \operatorname{argmax}\{d(s,u) \mid u \in U\},$$

and sort S by the descending order of $d(s, u_s^{\min})$.

Step 2. Find $Z_1 := \{s'\}$, where $s' \in \operatorname{argmin}\{\mathsf{delay}(\{s\}) \mid s \in S\}$.

Step 3. For each element $k \in D_1$ in descending order, do the following.

Step 3-1. Determine the set

$$Y^k := \{s \in S \mid d(s, u_s^{\min}) \leq k\}$$

by linear search over S, and sort Y^k in the descending order of $d(s, u_s^{\max})$. Let $Y^k = \{s_1, s_2, \ldots, s_{t(k)}\}$ in that order.

Step 3-2. For each $i \in \{1, 2, \ldots, t(k)\}$, determine

$$\ell_i := \min\{\ell \in D_2 \mid d(s_i, u_{s_i}^{\max}) \leq \ell + k\}$$

by binary search over D_2, and determine $X^{k,\ell_i} := \{s \in Y^k \mid d(s, u_s^{\max}) \leq k + \ell_i\}$ by linear search over Y^k.

Step 4. Find $\min\{\mathsf{delay}(X^{k,\ell_i}) \mid k \in D_1, i \in \{1, 2, \ldots, t(k)\}\}$, and let Z_2 be the set X^{k,ℓ_i} attaining the minimum.

Step 5. If $\mathsf{delay}(Z_1) < \mathsf{delay}(Z_2)$, then return $Z := Z_1$, and otherwise, return $Z := Z_2$.

For Case 1, an easy but useful observation is the following. For a server $s \in S$, let

$$u_s^{\min} \in \operatorname{argmin}\{d(s,u) \mid u \in U\} \quad \text{and} \quad u_s^{\max} \in \operatorname{argmax}\{d(s,u) \mid u \in U\}.$$

Namely, $u_s^{\min}$ is the closest user to s and $u_s^{\max}$ is the furthest user from s. Then, $s \in X^{k,\ell}$ if and only if $d(s, u_s^{\min}) \leq k$ and $d(s, u_s^{\max}) \leq k + \ell$. This observation makes the determination of $X^{k,\ell}$ easier. Furthermore, the number of different $X^{k,\ell}$ is small: it can be bounded by $O(|D_1||D_2|)$, but can be reduced to $O(|S|^2)$. Those observations result in speed-up of the algorithm.

The modified algorithm for Case 1 is described as Algorithm 3. We will see that it runs in $O(|S|^2(|S| + |U|))$ time. In fact, Step 1-1 takes $O(|S|(|S| + |U|) + |D_1| \log |D_1| + |D_2| \log |D_2|) = O(|S||U|(\log |S| + \log |U|) + |S|^2 \log |S|)$ time. Step 1-2 takes $O(|S||U| + |S| \log |S|)$ time. Step 2 takes $O(|S||U|)$ time as Algorithm 1. In Step 3-1, the linear search takes $O(|D_1| + |S|)$ time, and the sorting takes $O(|S| \log |S|)$ time by insertion sort in total since we only have at most $|S|$ distinct sets as Y^k. Thus, Step 3-1 takes $O(|S|(|U| + \log |S|))$ time. Step 3-2 takes $O(|Y^k| \log |D_2|) = O(|S| \log |S|)$ time. Step 4 can be executed in $O(|S|^2(|S|+|U|))$ time since we only have at most $|S|$ distinct sets as Y^k, and for each distinct Y^k we scan the relevant entries of d (the number of relevant entries is $O(|S|(|S| + |U|))$. Step 5 takes $O(1)$ time. Therefore, in total, Algorithm 3 runs in $O(|S|^2(|S| + |U|))$ time.

Algorithm 4 Accelerated Algorithm 2

Step 1-1. Compute D_2 and sort D_2 in descending order.

Step 1-2. For each server s in S, define

$$D_{2,s} := \{d(s,t) \mid t \in S\},$$

and sort $D_{2,s}$ in decreasing order. Note that $D_2 = \bigcup_{s \in S} D_{2,s}$ as a set.

Step 2. For each server s in S and each element ℓ in $D_{2,s}$ in decreasing order, determine

$$X_s^\ell := \{t \in S \mid d(s,t) \leq \ell\}$$

by linear search, and find $\min\{\mathsf{delay}(X_s^\ell) \mid s \in S\}$, and let X^ℓ be the set attaining the minimum.

Step 3. Find $\min\{\mathsf{delay}(X^\ell) \mid \ell \in D_2\}$, and return the set attaining the minimum as Z_3.

For Case 2, we obtain a similar speed-up, as described in Algorithm 4. The algorithm runs in $O(|S|^2 \log |S| + |S||U|)$ time, which is $\tilde{O}(|S|(|S| + |U|))$. More specifically, Step 1-1 takes $O(|D_2| \log |D_2|) = O(|S|^2 \log |S|)$ time, and Step 1-2 takes $O(|S|^2 \log |S|)$ time in total. In Step 2, with the help of sorted orders, the computation takes $O(|S|(|S| + |U|))$ time. Step 3 takes $O(|D_2|) = O(|S|^2)$ time. Therefore, in total, Algorithm 4 runs in $O(|S|^2 \log |S| + |S||U|)$ time, which is $\tilde{O}(|S|(|S| + |U|))$.

Note that in Algorithms 3 and 4, we do not make use of the experiment setup where the instances are given on the Euclidean plane. They can run on any instances of the problem.

As a preliminary experiment showed, Algorithm 3 was still slow in practice. Thus, we consider a heuristic improvement of running time by losing the theoretical guarantee for the approximation ratio. The idea is a simple interleaving. After we construct the set D_1, we throw most of the elements in D_1 away, but we only keep at most $\sqrt{|D_1|}$ elements. Namely, we only keep every $\sqrt{|D_1|}$-th elements after the set is sorted. The speed-up is gained by replacing every occurrence of $|D_1|$ by $\sqrt{|D_1|}$, in the running time analysis after sorting D_1. Then, we conclude that the heuristic interleaving of Algorithm 3 results in an algorithm that yet runs in $O(|S|^2(|S| + |U|))$ time, but practically it is much faster than Algorithm 3 as we will see.

In the following experiments, algo1 refers to the accelerated version of Algorithm 1 (thus, Algorithm 3), algo2 refers to the accelerated version of Algorithm 2 (thus, Algorithm 4), algo1-red refers to the accelerated version of Algorithm 1 with the heuristic interleaving, and CPLEX refers to the ILP formulation solved by CPLEX 12.7.0.0.

4.2 Experiment 1: Following Kawabata *et al.* [11]

In Experiment 1, we follow the experiment setup by Kawabata *et al.* [11]. The users are distributed uniformly at random in the square $[0, 20] \times [0, 20]$. The

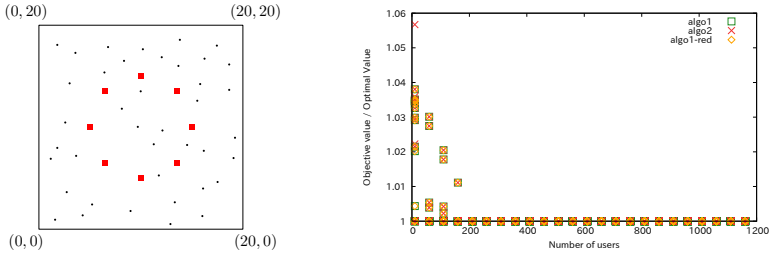

Fig. 1. (Left) The setup of Experiment 1. Red squares are servers, and black dots are users. (Right) Comparison of objective values in Experiment 1.

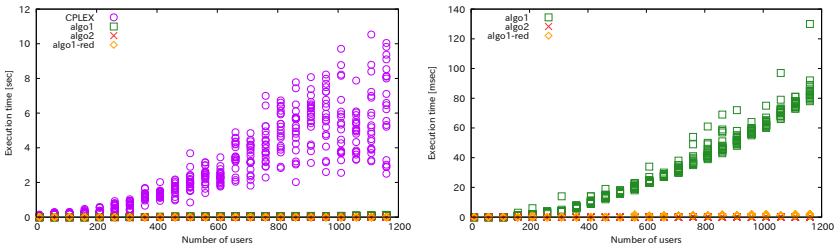

Fig. 2. (Left) Comparison of execution times in Experiment 1. (Right) The same plot as left, but the CPLEX data are dropped.

number of servers is fixed to eight, and they are placed at the position of vertices of a regular octagon, which is centered at $(10, 10)$ and has diameter 10. The number of users ranges from 10 to 960 with increment of 50. For each number of users, 20 data are generated. Delay is simply defined by the Euclidean distance, and thus is metric in $S \cup U$. See Fig. 1 (left).

Figure 1 (right) shows the comparison of objective values. The vertical axis refers to the approximation ratio, where the optimal values are computed by CPLEX. When the number of users is small (up to 160), the proposed algorithms sometimes output suboptimal solutions, but when the number of users is larger, they always output optimal solutions. Even in the worst case, the approximation ratio is less than 1.06.

Figure 2 compares the execution times. As the left figure shows, the proposed algorithms run much faster than CPLEX. The right figure shows the comparison between the proposed algorithms, without CPLEX. We can see that algo1 is slower than algo2 and algo1-red that have the comparable performance.[6]

As a summary of Experiment 1, we conclude that the proposed algorithms scale better than CPLEX, and the solution quality is not as bad as the theoretical guarantees predict.

[6] The reader may wonder why the authors did not make a semi-log plot, which could show the trend better. However, this was impossible since some instances were solved in "0 milliseconds," and taking the logarithm produced $-\infty$.

Table 2. The number of instances that were solved within the time limit of 120 seconds.

$\|U\|$	100	250	400	550	700	850	1000
CPLEX	38	32	31	27	24	23	22
algo1	56	56	56	55	52	47	43
algo2	56	56	56	56	56	56	56
algo1-red	56	56	56	56	56	56	56

$\|S\|$	1	5	9	13	17	21	25	29	50	80	110	140	170	200
CPLEX	28	28	28	28	28	25	15	11	4	2	0	0	0	0
algo1	28	28	28	28	28	28	28	28	28	28	27	23	20	15
algo2	28	28	28	28	28	28	28	28	28	28	28	28	28	28
algo1-red	28	28	28	28	28	28	28	28	28	28	28	28	28	28

4.3 Experiment 2: With More Servers

In Experiment 2, we again follow the experiment setup by Kawabata *et al.* [11] as Experiment 1, but the number of servers will be changed. The servers and users are distributed uniformly at random in the square $[0, 20] \times [0, 20]$. The number of servers ranges from 1 to 29 with increment of 4, and from 50 to 200 with increment of 30. The number of users ranges from 100 from 1000 with increment of 150. For each number of servers and users, four data are generated. Delay is simply defined by the Euclidean distance, and thus is metric in $S \cup U$. We set the time limit to 120 seconds for all instances.

Table 2 shows the number of instances that were solved within the time limit of 120 seconds. Here, "solved" means that the algorithm successfully terminates. The upper table is the summary as the number of users varies. For each entry, the total number of generated instances is 56. As it shows, algo2 and algo1-red solve all the instances, while CPLEX cannot solve some instances with 100 users. The lower table is the summary as the number of servers varies. Again, algo2 and algo1-red solve all the instances, which is also a consequence of the upper table, but CPLEX cannot solve any instances with 110 or more servers. We also see that algo1 cannot solve some instances, and thus we conclude that the heuristic interleaving works pretty well.

Figure 3 compares the execution times. The left figure corresponds to the case with 50 servers, the right figure corresponds to the case with 200 servers, and the horizontal axis refers to the number of users. As they show, algo2 and algo1-red scale very well, and even when the number of users is 1000, they run within ten milliseconds.

Figure 4 compares the objective values. The plot is restricted to the instances that CPLEX was able to solve since we need to know the optimal value for plotting the approximation ratio. The left figure corresponds to the case with 25 servers, where the horizontal axis refers to the number of users. The right figure corresponds to the case with 1000 users, where the horizontal axis refers to the number of servers. As they show, algo2 tends to output worse solutions than algo1 and algo1-red, but the approximation ratio never exceeds 1.15. Our inspection

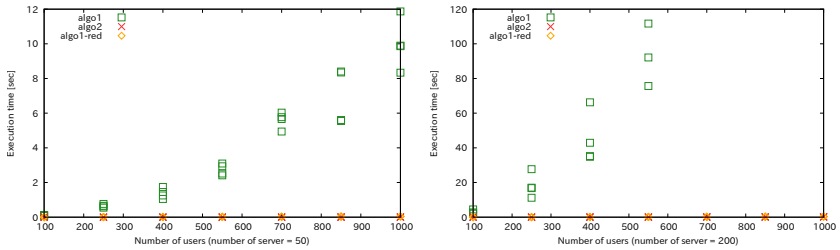

Fig. 3. (Left) Comparison of execution times in Experiment 2, for the case with 50 servers. (Right) The same plot with 200 servers.

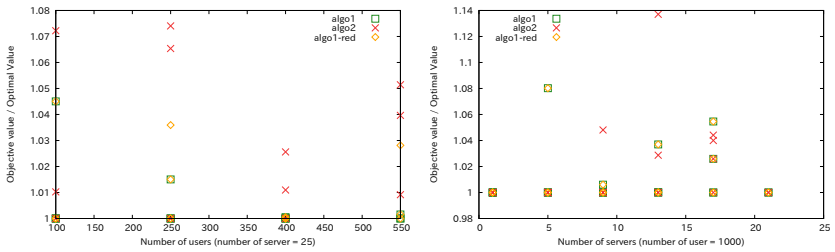

Fig. 4. (Left) Comparison of objective values in Experiment 2, for the case with 25 servers with different numbers of users. (Right) The same plot with 1000 users with different numbers of servers. Each point corresponds to a single instance that was solved by CPLEX.

shows that the approximation ratios stay below 1.19 for all the instances that were solved by CPLEX.

Acknowledgments We thank Eiji Oki for bringing the problem into our attention.

References

1. Sibel Alumur and Bahar Y Kara. Network hub location problems: The state of the art. *European Journal of Operational Research*, 190(1):1–21, 2008.
2. S. Ba, A. Kawabata, B. C. Chatterjee, and E. Oki. Computational time complexity of allocation problem for distributed servers in real-time applications. In *Proceedings of 18th Asia-Pacific Network Operations and Management Symposium*, pages 1–4, 2016.
3. J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–756, 2015.
4. James F Campbell. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72(2):387–405, 1994.

5. Li-Hsuan Chen, Dun-Wei Cheng, Sun-Yuan Hsieh, Ling-Ju Hung, Chia-Wei Lee, and Bang Ye Wu. Approximation algorithms for single allocation k-hub center problem. In *Proceedings of the 33rd Workshop on Combinatorial Mathematics and Computation Theory (CMCT 2016)*, pages 13–18, 2016.
6. Reza Zanjirani Farahani, Masoud Hekmatfar, Alireza Boloori Arabani, and Ehsan Nikbakhsh. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, 64(4):1096–1109, 2013.
7. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
8. D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.
9. R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
10. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
11. A. Kawabata, B. C. Chatterjee, and E. Oki. Distributed processing communication scheme for real-time applications considering admissible delay. In *Proceedings of 2016 IEEE International Workshop Technical Committee on Communications Quality and Reliability*, pages 1–6, 2016.
12. Morton E O'Kelly and Harvey J Miller. Solution strategies for the single facility minimax hub location problem. *Papers in Regional Science*, 70(4):367–380, 1991.