Data Centers Selection for Moving Geo-distributed Big Data to Cloud

Jiangtao Zhang[1], Qiang Yuan[2,3], Shi Chen[2,3], Hejiao Huang[2,3], Xuan Wang[2,4]

[1] Shenzhen Jingyi Smart Technology Co., Ltd., China
[2] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, China
[3] Shenzhen Key Laboratory of Internet of Information Collaboration, China
[4] Shenzhen Applied Technology Engineering Laboratory for Internet Multimedia Application, China
jiangtaozhang@aliyun.com, 976796147@qq.com, honesty_chenshi@163.com,
huanghejiao@hit.edu.cn, wangxuan@insun.hit.edu.cn

Abstract

Because of the distributed networking and coexistent abundant computation and storage resources, cloud computing has become a preferred platform for big data analytics, especially for the geo-distributed data across the world. The precondition for data processing is to move the data to the cloud. Due to the large volume of data, high transmission cost across continents and even specific legal prohibition, it is not always feasible to move all data to one data center. Appropriate data centers should be selected while keeping fast data access and low cost. In this paper, four criteria of the problem are explored. A tight 3-approximation algorithm is proposed to address the former two criteria. It can be simplified when the underlying bipartite graph is complete. The latter two criteria are addressed by a heuristic. Comparing to the optimal method and other schemes, extensive simulations demonstrate that the proposed algorithms can find rather good solutions with less time, and hence are more appropriate for large scale applications.

Keywords: Big data, Data centers selection, Distributed cloud computing, Cost minimization

1 Introduction

Cloud computing has become a preferred platform for big data (BD) analytics [1-3], especially when data are produced from geo-distributed locations, where the local data are accessed often by local users, and sometimes the data need to be aggregated for further analysis or data mining [4]. For example, for a multi-national sales corporation consisting of a large number of branch companies across the world, branches in each country need to analyze the native customer related data in time for commercial purpose. All data should also be analyzed to report to headquarters or be joint-analyzed to facilitate the cross-border deals. Normally, a large scale cloud is networked in a distributed fashion and has multiple geo-s panned data centers (DCs, at least 16 DCs spanning 4 continents for Amazon [5] and 13 DCs across 4 continents for Google [6]). Each DC is equipped with computation integrated with storage resources in an elastic pay-as-you-go mode. This infrastructure can provide nearby service and is particularly appropriate for geo-distributed data processing. To process BD in cloud, the precondition is to move and save BD in eligible DCs [2]. Hard-driven mechanism is one option to move the large volume data. For example, Amazon Import/Export service recommends using portable storage devices to ship data [7]. Sometimes, it is even possible to move the whole machine [8]. But this usually applies to intermittent or one-time bulk data moving. It invokes great delay and cannot meet the increasing real-time analysis requirement [1]. Moreover, it contradicts automated administration and needs more labor, which is continually becoming more expensive. Data transmission over the Internet is costly and unrealistic due to the great delay. For example, it will take about 13 days to transfer 1TB data via a 10MB internet connection [7]. Dedicated high speed connection is often suggested to move the real-time data (e.g., AWS direct connect of Amazon [9]). This method can facilitate faster transmission. But even relying on the high speed dedicated connection, moving BD across continents is still very difficult. For example, AWS direct connect does not provide across continents service. The international private leased circuit is very expensive. This hampers moving large scale data, which disperse around the world, to only one DC [10]. Furthermore, using one DC to store data incurs a higher data access delay for the more frequent local data analysis. Especially in some regions, data security laws require some data must be stored locally (e.g., some countries in EU). Taken together, users should respect some criteria to select locations for their data,

*Corresponding Author: Hejiao Huang; E-mail: huanghejiao@hit.edu.cn
DOI: 10.3966/160792642019012001010

just as that recommended by Amazon: being nearer to users to reduce data access latency, addressing specific legal and regulatory requirements or reducing cost, etc.

Figure 1 models the distribution of DCs and users (we use users to represent the branch companies or a group of data owners for simplicity). There are 10 DCs and 9 users across 4 continents. Different DCs are available for or preferred by different users. For example, users 4, 5 in Europe can only be assigned to DCs 5, 6 in the same continent. Users in Asia only prefer DCs in the Asian-Pacific region.

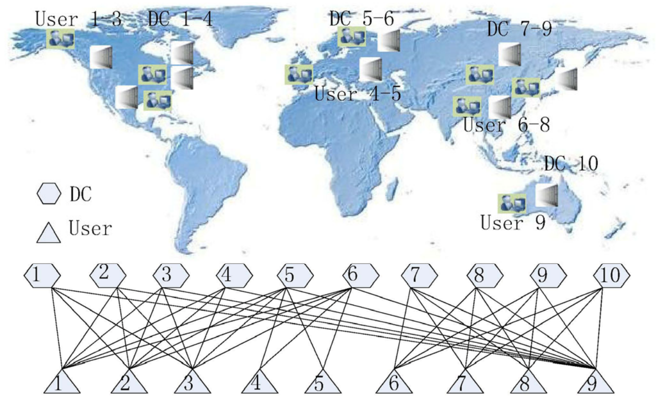


Figure 1. Geo-distributed data centers and users. The line in the underlying bipartite graph indicates the availability of DCs or the preference of users.

Nowadays, some MapReduce-like framework, such as GHadoop [11], G-MR [12], G-framework [13] and Cross-cloud MapReduce [14], can realize data analysis across clusters and DCs [10, 15]. Comparing with one DC scheme, multiple DCs mechanism can not only meet the aggregate analysis requirement, but also guarantee faster access and lower cost.

This work explores the target DCs selection problem. The main contributions are as follows:

1. We formalize the target DCs selection problem with four criteria: fair data placement (FDP), preferential data placement (PDP), transmission cost minimization data placement (TCMDP) and cost minimization data placement (CMDP).

2. Considering the underlying incomplete bipartite graph, which can reflect the availability of DCs and the preference of users in practice, a tight 3-approximation algorithm is proposed for the first two criteria. The latter two are solved by a simple heuristic. If the graph is complete, the approximation algorithm can be simplified further.

3. Extensive simulations demonstrate the effectiveness of the proposed algorithms. They can find rather good solutions with less time.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 formulates the problem and Section 4 presents algorithms to address different criteria. Section 5 evaluates the algorithms and Section 6 concludes the work.

2 Related Work

2.1 Moving Big Data to Cloud

Few literatures explore moving big data to cloud [16-17]. Focusing on dynamic big data, the authors of [16, 18] prefer to select a most suited DC to store data while optimizing monetary cost. Two online algorithms are proposed to seek the target DC. Due to the dynamic data, the best target DC changes continually and data need to be moved accordingly. It is assumed that only the recent part of data needs to be migrated between DCs. Without this assumption, it is obvious that the accumulated historical data transfer cost will be catastrophic (Based on TDWI report [19], one third of organizations have already broken 10TB barrier in 2011.). Nevertheless, this method will cause the user's data to spread over a large number of DCs after a rather long period and therefore very hard to manage. Genomics data delivery and processing are explored by a proposed decentralized architecture [17]. But all the data also need to be aggregated to one DC. In both works, it is assumed implicitly that there exists at least one DC available for all data across the world. It is not true sometimes due to the legal prohibition or other factors.

Social data and replica are explored to move to multi-cloud considering users' social relation and their potential interaction [20]. The problem is formulated as a multi-objective program and solved by a heuristic based on graph cut. It also demonstrates the necessity of storing distributed data in different locations. Other papers discuss electricity cost and carbon emission [21] or system throughout [22], but the transmission cost is not considered.

2.2 Facility Location Problem and *k*-median Problem

Multiple DCs selection for moving BD to cloud is related to facility location problem (FLP) and *k*-median problem. FLP aims to find facilities to serve clients based on various criteria. DCs can be viewed as facilities and local data users are just the clients. *k*-median problem strives to find a set of points not more than k . Each other point not be selected is assigned to one selected point so as to minimize the sum of length between the point pair.

In the variants of FLP, *k*-supplier problem is one of the most related ones to our problem PDP, in that at most k suppliers (correspond to DCs) need to be selected from a given set so that the maximum distance between each client and its closest supplier is minimized [23]. Normally, the suppliers and clients network is modeled as a complete graph [23] or a complete bipartite graph [24]. While in our problem, prohibited by laws, maybe some DCs cannot be selected to serve some data. So the graph is not always a complete graph. For a generalized *k*-supplier variant where each supplier is attached with a weight, it is required the total suppliers weight should not be bigger than k [24]. A 3-approximation algorithm is presented and it is proved as tight (i.e., there is no smaller approximation ratio). Normally, it cannot apply to our problem where the data to be moved is attached with users, not with DCs (suppliers). Usage weight is studied in another variant: *k*-center problem [25]. *k*-center problem differs in essence from *k*-suppler problem in that the former requires all the points which are not selected as centers must be close to one center, while *k*-supplier problem does not require the suppliers not selected to be close to any of the suppliers selected. The corresponding schemes cannot be used to address our problem too.

Our problem CMDP is a common generalization of uncapacitated facility location (UFL) problem and *k*-median problem. UFL is another variant of FLP where there is no capacity limitation for the facility and each facility is attached with a fixed open cost weight [26]. The objective tries to minimize the total fixed cost and the total service cost. It can be explored in metric space and non-metric space. The metric UFL is strongly NP-hard and cannot be approximated with a ratio smaller than 1.436 unless $NP \subseteq DTIME[O^{loglogn}]$ or $P = NP$ [27]. Few works explore non-metric UFL [26, 28]. They obtain an approximation factor with a constraint violation. The authors of [26] present a filter and rounding mechanism. A $1+\varepsilon$ approximation factor is achieved with a violation factor of $1+1/\varepsilon(lnn+1)$, that means the algorithm needs a facility subset of size not less than $k(1+1/\varepsilon(lnn+1))$. The same approximation factor is derived with a violation factor of $lnn + n/\varepsilon$ by an oblivious rounding and greedy method [28]. Obviously this violation cannot be tolerant for our CMDP because the number of DCs is relatively small. Moreover, the fixed facility weight differs from CMDP in that DC weight is variable and is proportional to usage weight. Another difference is that UFL does not limit the number of facilities.

k-median problem is explored in [29]. A 1-swap algorithm is presented and it gives a 5-approximation solution. It is refined to a p-swap algorithm which can guarantee a $3+2/p$ approximation factor. But here all the cost are measured by the distance between clients and facilities. *k*-median problem does not consider the weights of facilities.

So our CMDP generalizes UFL and *k*-median problems simultaneously. Noting that in all the above works, the underlying graph is assumed to be complete. CMDP extends the situation to incomplete graph based on the practical requirement in moving big data to cloud.

3 Problem Description and Formulation

Considering the underlying incomplete bipartite graph $G=(U,V,E)$ whose edge $e_{ij} \in E(i \in U, j \in V)$ satisfies the triangle inequality and a positive integer k $(k \le |U|, k \le |V|)$, we try to find a DC subset D where $|D| \le k$ from the available DC set V to store data of all users in U with different criteria. For any $i \in U$ and $j \in V$, there is an edge between them if data of user i can be moved to DC j (at least it is not prohibited by law or is not excluded by the user). Assume any i is adjacent to at least one j, otherwise there is no solution. Suppose $|E| = m$ where $m \le |U| * |V|$. In this paper we also use e_{ij} to indicate the length of edge e_{ij} if there is no confusion.

Usage weight. Each user is attached with a weight w_i which denotes the activity level of data production of current or foreseeable future days, or the importance of the local user. w_i increases when the data volume or the importance increases. Using activity level instead of the data volume can tolerate the dynamic fluctuation of data while giving an approximation of data volume. Active level can be defined based on the daily volume uploaded. Such as for a typical corporation with 200 GB upload every day [19], 10 GB can be used as the threshold to judge the active level. If a branch corporation produces data smaller than 10 GB, it is attached with a weight 1. For a branch with data between 20~30 GB, the weight is 3, and so on. For user i with active level w_i, it needs to pay $w_i e_{ij}$ to the cloud if it wants to move data to DC j.

DC weight. Each DC differs in computation and storage pricing (e.g., various Amazon VM instances and S3 [30] pricing in different regions). To store and process data economically, lower price is preferred. Given DC j, suppose that the price of a VM instance to process data is a_j every hour and the instance can analyze b_j GB data each hour in average, then the price to process 10 GB is $p_j^{'} = 10/b_j * a_j$. If the storage price for 10GB is $p_j^{''}$, then the total cost of DC side is $p_j = p_j^{'} + p_j^{''}$ for user with activity level 1. For user i with active level w_i, it needs to pay extra $w_i p_j$ if it wants to store and process data in DC j. The overall cost of user with w_i is $w_i(p_j + e_{ij})$. Considering the order-of-magnitude difference for w_i (e.g., thousands of kilometers) and p_j (e.g., about several dollars one hour for Amazon) in practical environment, a normalized variant is used: $c_{ij} = w_i(p_j^{'''} + e_{ij}^{'})$, where $p_j^{'''} = p_j / \max_{h \in V}(p_h)$, $e_{ij}^{'} = e_{ij} / \max_{l \in U, h \in V}(e_{lh})$. Note that the normalized edge lengths $e_{ij}^{'}$ still satisfy the triangle inequality.

The main notations used in the formulation are summarized in Table 1.

Table 1. notations

Notation	Description
$G=(U,V,E)$	Bipartite Graph G with user vertex set U, DC vertex set V and edge set E
e_{ij}	Edge between vertex i and j or the length of edge e_{ij}
$\|U\|$	Number of vertices in vertices set U
$\|E\|$	Number of edges in edge set E
k	A positive integer k $(k \le \|U\|, k \le \|V\|)$
D	A subset of DC vertices set V where $\|D\| \le k$
m	Number of edges in edge set E
w_i	Usage weight, denote the activity level of data production
a_j	The price of a VM instance in DC j to process data every hour
b_j	The data volume (GB) can be analyzed by the VM instance in DC j
$p_j^{'}$	The price to process 10 GB in DC j
$p_j^{''}$	The storage price for 10GB in DC j
p_j	The total cost of DC side for user with activity level 1
c_{ij}	The normalized overall cost of user with w_i

Problem description and formulation. Due to the high scalability of cloud, we assume that there is no computation and storage capacity limitation for DCs. The data of one user are only stored in one DC for data integrity. The target data centers selection problem can be summarized as: select at most k target DCs and assign each user to one target DC to place data, so as to meet the following criteria.

1. Fair data placement (FDP). The maximum distance between each user and its assigned DC is minimized so that each local user can access data with minimal latency: $\min_{D\subseteq V,|D|\le k}\max_{i\in U,j\in D}(e_{ij})$.

2. Preferential data placement (PDP). The maximum usage weighted distance between each user and its assigned DC is minimized so that local users with more data can access data with minimal latency: $\min_{D\subseteq V,|D|\le k}\max_{i\in U,j\in D}(e_{ij})$. We use $w(i,j) = w_i e_{ij}$ to denote the weighted distance if necessary.

3. Transmission cost minimization data placement (TCMDP). The transmission cost, defined as the sum of the usage 3 weighted distance between each user and its assigned DC, is minimized: $\min_{D\subseteq V,|D|\le k}(\sum_{i\in U,j\in D} w_i e_{ij})$.

4. Cost minimization data placement (CMDP). The overall cost, defined as the sum of the cost of each user is minimized: $\min_{D\subseteq V,|D|\le k}(\sum_{i\in U,j\in D} c_{ij})$.

Note that criterion (1) is a special case of criterion (2) when $w_i = 1$, and criterion (3) is a special case of criterion (4) when $p_j = 0$, we mainly talk about (2) and (4). The corresponding results can apply directly to (1) and (3), respectively.

Let x_{ij} is a boolean variable which indicates whether user i is assigned to DC j. It equals 1 if i is assigned to j and 0 otherwise. y_j is also a boolean variable which indicates whether DC j is used. It equals 1 if j is used and 0 otherwise.

For PDP problem, we want to minimize the maximum weighted distance z, where

$$w_i e_{ij} x_{ij} \le z \ \forall i \in U, j \in V \qquad \textbf{(1)}$$

PDP can be formulated as the following mixed 0-1 integer linear program:

min z

s.t. **(1)**

$$\sum_{j\in V} x_{ij} = 1 \qquad \forall i \in U \qquad \textbf{(2)}$$

$$y_j \ge x_{ij} \qquad \forall i \in U, j \in V \qquad \textbf{(3)}$$

$$\sum_{j\in V} y_j \le k \qquad \textbf{(4)}$$

$$x_{ij} \in \{0,1\}, y_j \in \{0,1\} \qquad \forall i \in U, j \in V \qquad \textbf{(5)}$$

$$x_{ij} = 0 \qquad \textit{for some } i \in U, j \in V \qquad \textbf{(6)}$$

Constraint (2) ensures that each user must be assigned to at least 1 DC and constraint (3) ensures this DC must be used, i.e., at least one VM is assigned to the DC. The number of DCs used cannot exceed k (4). Furthermore, not all DCs are available for each user (6), i.e., the underlying bipartite graph is incomplete.

CMDP can be formalized as the following 0-1 integer linear program:

$$\min \sum_{i\in U,j\in V}(w_i e_{ij} + w_i p_j) x_{ij}$$

s.t. (2), (3), (4), (5), (6).

In the objective function, e_{ij} and p_j are all the normalized ones. The former part $\sum_{i\in U,j\in V} w_i e_{ij} x_{ij}$ is the transmission cost, the latter part $\sum_{i\in U,j\in V} w_i p_j x_{ij}$ is the data processing and storage cost in DC.

As stated in Section 2, PDP is an extension of *k*-supplier problem, and CMDP is a common extension of UFL and kmedian problem. Because *k*-supplier problem, UFL and kmedian problem are all NP-hard [26, 31], PDP and CMDP are both NP-hard. An approximation algorithm is presented for PDP and a heuristic is proposed for CMDP.

4 Algorithm

Some notions are introduced to facilitate the algorithm presentation.

Bottleneck graph. Note that for PDP problem, the optimal solution must reach at one usage weighted edge, so we should check the weighted edge from the smallest to the biggest until all constraints are satisfied. The construction of bottleneck graph is just based on this idea. m usage weighted edges are sorted in a non-decreasing order and denote them as $w(1,j) \le w(2,j) \le \ldots w(m,h)$ where $j,g,h \in V$ and may be the same. Bottleneck graphs $G_1,G_2,...,G_m$ are edge subgraphs of G and $(r=1,2,...,m)$ where $G_r=(U,V,E_r)$, $E_r=\{e_{ij} \mid w_i e_{ij} \le w(r,g)\}$. Namely, G_r consists of all vertices of G and edges which is not bigger than the r-th shortest weighted edge $w(r,g)$.

Threshold graph. The threshold graph T_r is constructed on user set U for each G_r as follows. For each two points $u,v \in U$, an edge (u, v) is in T_r if and only if there exists a DC $j \in V$ with both (u,j) and (v,j) in G_r. For example, Figure 2 illustrates the threshold graph T_m for T_m (the biggest bottleneck graph) which is just the bipartite graph in Figure 1. There is no edge (5,6) because there is no common adjacent DC to users 5, 6 in G_m.

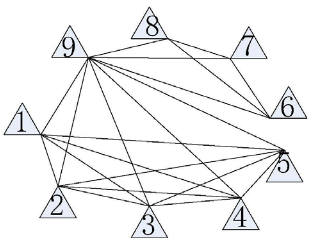


Figure 2. The threshold graph T_m

Maximal clique. Given an undirected graph H, a clique of H is a complete subgraph. If a clique is not contained in any other clique, it is a maximal clique and is denoted as $C(H)$. It is easy to find $C(H)$ in a polynomial time. A simple method used in this paper is to add any point of H to $C(H)$ first, then add its neighbor points, together with which all points in $C(H)$ can constitute a complete subgraph. All neighbors of the points in $C(H)$ are checked one by one until no point can be added. A maximal clique can be found. Now, delete the points in this clique and repeat this process in the residue points until H becomes null. A series of maximal cliques can be found and they have no intersection.

Note that if users can be served by a common DC then the corresponding threshold graph is a clique. For example, in Figure 1, users 6~9 can be served by common DCs then the corresponding threshold subgraph in T_m is the clique consisting of 6~9 in Figure 2. Finding the maximal cliques in the threshold graph means grouping the users based on the availability of DCs so that all users in the group can be represented by a certain member.

4.1 Preferential Data Placement (PDP)

4.1.1 For the Scenario of Incomplete Graph

Let $N(i)$ denote the neighbors of i, i.e., points which are adjacent to i. The algorithm is detailed in Alg. 1.

Algorithm 1. Biggest Weight First (BWF)

```
Input: $G=(U,V,E)$: usage weighted incomplete
        bipartite graph
        $k$: positive integer
Output: $D$: set of data centers selected
1:  $D \leftarrow \varnothing$
2:  Sort usage weighted edges in a non-decreasing
    order: $w(1,j) \le w(2,g) \le \ldots \le w(m,h)$
3:  Construct threshold graph $T_r$ for each $G(r)$. Find
    the maximal clique set for each $T_r$. Suppose there
    are $H$ cliques. Denote the clique set for $T_r$
    is $C_r = \{C(T_r)_h\}$ where $r=1,2,\ldots,m, h=1,\ldots,H$
4:  Let $t = min\{r \,\|\, C_r \,| \le k\}$
5:  for $h=1,\ldots,H$ do
6:  Let $l$ is the point with $w_l = max\{w_j \mid j \in C(T_t)_h, j$ is
    not assigned$\}$. Let the common neighbor DCs of
    $C(T_t)h$ is $N \leftarrow \bigcap_{u \in C(T_t)h}(N(u) \subseteq V)$
7:  $e \leftarrow min_{j \in N} \{e_{lj} \mid e_{lj} \in G_t\}$
8:  Let $v$ is the point in $N$ with $e_{lv} = e$
9:  $D \leftarrow D \cup v$
10: end for
```

Algorithm 1 tries to place the largest volume data to the nearest DC. It checks the bottleneck graph in terms of usage weighted edge from small to big one by one and constructs the corresponding threshold graph. Then the clique set for each threshold graph is found (Line 2~Line 3). Each clique represents a user group which can be served by at least one common DC and the weighted edges between users and the common DCs are bounded by the maximal weighted edge of the bottleneck graph. The minimum index of all clique sets which cardinalities are not bigger than k is selected so that the biggest usage weighted edge is as small as possible (Line 4). Of course binary search can be used to speed the search.

For each user in the threshold clique set C_t, we find the target DC as following. In each clique, beginning from the user with the biggest weight (Line 6), we uses it as a representative and assign it to the nearest neighbor DC in G_t (Line 7~Line 9). All other users in the same clique are also assigned to this DC implicitly. This process is repeated until all users in C_t are assigned (the **for** loop). Thus the algorithm establishes a mapping between users and DCs. The mapping partitions the bipartite graph into multiple clusters and the center of each cluster is a DC.

The following theorem demonstrates that Alg. 1 finds a tight 3-approximation solution for PDP problem where the underlying bipartite graph is incomplete.

Theorem 4.1. *Algorithm 1 gives a 3-approximation solution for PDP problem where the underlying bipartite graph is incomplete and the solution is tight.*

Proof. The optimal value OPT must be reached at one edge. Let t^* is the optimal edge index of the bottleneck graph G_{t*}. Because t^* also satisfies $|C_t^*| \le k$, algorithm 1 finds the minimum t (Line 4), so $t \le t^*$. We get $w(t,j) \le w(t^*, l) = OPT$ based on Line 2 in Alg. 1.

Note $|D| \le k$ and $w(t,j) \le OPT$, we only need to prove for any $u \in U$, there is a DC $v \in V$ which can cover u with $3w(t,j)$.

As shown in Figure 3, for any $u \in U$, since C_t is the maximal clique set of G_t and C_t covers V, so u must be in a clique in C_t, such as $C(T_t)_i$. If u is a maximal clique itself or $w(u)$ is the maximum weight in $C(T_t)_i$, u is covered by it nearest DC in G_t within $w(t,j)$. We get it.

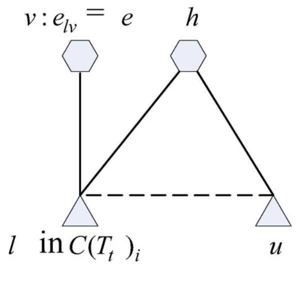


Figure 3. Diagram of theorem 4.1

Otherwise, there must exists l in $C(T_t)_i$ where $w(l) \ge w(u)$ so that e_{ul} in T_t. This means there exists a DC $h \in V$ so that edges e_{uh} and e_{lh} in G_t. So $w(u, h) \le w(t,j)$ and $w(l,h) \le w(t,j)$. Based on the algorithm and the definition of bottleneck graph, we also have $w(l, v) \le w(t,j)$.

$$\begin{aligned} w(u, v) &= w(u)e_{uv} \\ &\le w(u)(e_{uh} + e_{lh} + e_{lv}) \\ &\le w(u)e_{uh} + w(l)e_{lh} + w(l)e_{lv} \\ &= w(u, h) + w(l, h) + w(l, v) \\ &\le 3w(t, j). \end{aligned}$$

The first inequality is due to triangular inequality.

Suppose it is not tight, then there exists an algorithm which can find a smaller approximation ratio. It is also apply to the case when all usage weights are 1, which is just a special case of *k*-supplier problem where all supplier weights are 1. This contradicts the theorem that 3 is tight for *k*-supplier problem [23].

Time complexity. Sorting m edges needs $O(mLogm)$. Suppose there are n users and d DCs. In Line 3, constructing all G_r needs $O(m^2)$. Constructing all T_r needs $O(mlogn^2)$ and constructing all C_r needs at most $O(mn^2)$. It is dominated by $O(mn^2)$ and so Line 3 needs $O(mn^2)$ time. Note that the for loop needs $O(n(n+m))$ which is still dominated by the time of Line 3. The time complexity of algorithm 1 is $O(mn^2)$.

4.1.2 For the Scenario of Complete Graph

When the underlying bipartite graph is complete, we can simplify the grouping method. It is unnecessary to find the maximal clique in Line 3 in Alg 1. It is sufficient to find the maximal independent set for each T_r because all DCs are available for each user now.

Maximal independent set. Given an undirected graph H, an independent set of H is a subset of vertices of H and in which no two different vertices share an edge in H. If an independent set is not contained in any other independent set, it is a maximal independent set and is denoted as $I(H)$. It is easy to find $I(H)$ in a polynomial time. The simple method used here is to add any point of H to $I(H)$, then delete this point and all points adjacent to this point from H. Repeat this process until H becomes null.

In Alg. 1, if the maximal clique is replaced by the maximal independent set and the other parts remain unchanged. We get a simplified algorithm for the scenario of complete graph. This algorithm is denoted as the biggest weight first algorithm based on maximal independent set (BWFInd). Similar to the proof of theorem 4.1, it is easy to get the following theorem.

Theorem 4.2. *Algorithm BWFInd gives a 3-approximati-on solution for PDP problem where the underlying bipartite graph is complete and the solution is tight.*

Note that it is faster to find the maximal independent set than to find the maximal clique, BWFInd is faster than BWF. But it is easy to check that BWFInd cannot apply to the scenario of incomplete bipartite graph.

4.2 Cost Minimization Data Placement (CMDP)

For problem CMDP, Algorithm 2 is proposed. It tries to assign each user to the nearest DC, i.e., the cost between the user and the DC is the smallest. If the number of DC is bigger than k, then certain DC is discarded and the corresponding users are reassigned. The process is repeated until the constraint is satisfied.

```
Algorithm 2. CMDP Nearest Preferred Algorithm
             (NPACMDP)
Input: $G=(U,V,E)$ : weighted incomplete bipartite
       graph
       $k$ : positive integer
Output: $D$ : set of data centers selected
1: Assign each user to the nearest DC (the distance
   between user and DC is defined as $w_i e_{ij} + w_i p_j$).
   Record the DC set as $D$
2: while $|D| > k$ do
3:    Find one DC, for which the maximum weighted
      distance between it and the users assigned to it is
      the minimum when comparing with that of other
      DCs in $D$. . Delete this DC from $D$ . The users
      which has been assigned to the deleted DC are
      reassigned to the nearest remaining DC in $D$ .
4: end while
```

5 Simulation Results

5.1 Simulation Setting

Amazon, Google and NSFNET T3 [32-33] own representative DC networks. All of them have a dozen of DCs and the DCs of the former two span different continents. Since the location information of the DCs of business corporation is often confidential, no real precise information is released so far. We select 14 cities from four continents. Most of them are consistent with that of Amazon and some of them are the same as that of NSFNET T3. We also randomly select 17 cities from five continents for users.

It is detailed in Table 2[1]. Each DC and each user are numbered for simple presentation. DCs 1~10 correspond to DCs in Figure 1. Users 1~9 correspond to users in the same figure. Other extra DCs and users will be added in the later scalability evaluation. The distance between user and DC is estimated by the latitude and longitude of the city where they locate in. It is Euclidean distance and thus satisfies the triangular inequality. We suppose the availability of DCs for users is consistent with that implied in Figure 1. Namely, users in America can only be assigned to DCs in America and Europe. While users in Oceania and Africa can be assigned to any DCs in all four continents.

Table 2. Distributed DCs and users

	America							Europe					Asia					Oceania	Africa
No.	1	3	3	4	11			5	6	12	13		7	8	9	14		10	
DCs	SP	SFO	Toronto	Atlanta	NY			Frankfurt	Paris	London	Dublin		SG	Tokyo	HK	Beijing		Sydney	
c	0.067	0.067	0.077	0.067	0.067			0.079	0.0796	0.073	0.073		0.098	0.096	0.096	0.098		0.093	
c-M	0.077	0.077	0.067	0.077				0.073	0.073				0.096	0.098	0.093			0.096	
s	0.03	0.033	0.033	0.03	0.03			0.0324	0.0324	0.03	0.03		0.03	0.033	0.033	0.03		0.033	
s-M	0.03	0.03	0.03	0.033				0.03	0.03					0.033	0.033	0.03		0.033	
No.	1	2	3	10	11	12	13	4	5	14	15	16	6	7	8	18	19	9	17
users	CHI	WDC	LAX	Houston	Boston	PHIL	RDJ	Rome	Milan	Munich	Madrid	Moscow	BKK	Osaka	Xian	ULA	ND	Oakland	Cairo
w	20	15	28	10	18	29	2	21	8	18	13	25	7	11	24	40	40	16	1
w-M	15	20	10					13	25				24	11	7			1	

The weights of users follow $U(1-40)$ and are listed in the line beginning with w. One level represents that daily 1~10 GB data need to be uploaded to DC after preprocessing. We borrow the computation price ($/Hour) (abbreviated as c) and storage price ($/G) (abbreviated as s) from Amazon m3.medium virtual machine instance with Linux operation system and S3 standard storage in various regions, respectively. Suppose the instance can process one active level data every hour. Each DC is randomly attached with a computation and a storage price borrowed. The details can be found in Table 2. All prices have been multiplied by 100 to match the table space. To reveal the effect of DC weight and usage weight on the solution, we also modify the values of lines beginning with w , c and s to the values of lines beginning with $w-M$, $c-M$ and $s-M$, respectively. Herein only the modified weights to be used, i.e., DCs 1~10 and users1~9, are listed. On the whole, the weights after modification are smaller than the original ones. We set all the bandwidth cost for 10 GB are 0.01$ for comparison. Note that the objective is to minimize the product of distance and user weight, this will drive the data to be transmitted to a relative close DC and thus minimize both the cost and the delay.

For the DCs and users in Figure 1, obviously at least 2 DCs are necessary if all users can be served. So unless explicitly noted in the simulation, we use a slightly bigger $k = 3$.

In the simulation, we focus on the general situation where the underlying bipartite graph is incomplete. The simulation environment is set up in a C language platform. The platform is running on a PC (Lenovo Think Centre M4350t-N020, Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz, 8G RAM).

5.2 Simulation Results for PDP

To evaluate the performance of the proposed algorithm BWF, additional three schemes are compared. (1) Random scheme (Random): each user is randomly assigned to an available DC while keeping the number of DCs is not bigger than k. (2) Optimal scheme (Opt.): since PDP is linear programming, we use lpsolve in Sourceforge [34] to solve the corresponding linear programming in Section 3 to get the optimal values. lpsolve is written in C language and has been wisdely used. (3) NPA-PDP: replace the overall cost $w_i e_{ij} + w_i p_j$ in NPA-CMDP (Alg. 2) with the weighted distance $w_i e_{ij}$.

5.2.1 Scalability Evaluation

To investigate the DC scale efficiency of BWF, we suppose users 1 ~ 9 can be served by different number of candidate DCs 1~10. Every time one DC is added to DCs in the sequence of number from 11 to 14. The delay of each algorithm is depicted in Figure 5.

The delay (the maximum usage weighted distance) of Random is the biggest and the delays of NPA-PDP and BWF are bigger than that of Opt.. At beginning, both NPA-PDP and BWF select DCs 2, 6, 9. When the number of DCs increases to 13, the solutions of both NPA-PDP and BWF are the same. This is because that all the later added 3 DCs are not better than the former selected DCs, and therefore all of them are not selected. When DC 14 (Beijing) is added, the delay increases for both NPA-PDP and BWF. This attributes to that user 8

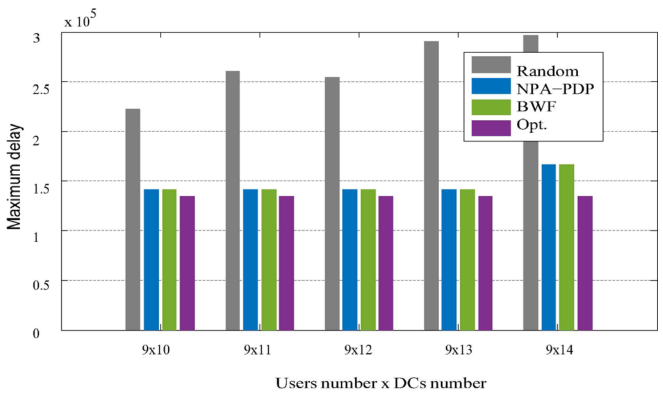


Figure 5. Maximum delay when the number of DC increases

(Xian) has the biggest usage weight and it is selected as the representative of users in Asia-Pacific region by the algorithms. User 8 is the nearest to DC 14, so DC 9 is given up and DC 14 is selected. Now users which are originally assigned to DC 9 are reassigned to DC 14. This leads to the delay increasing to the weighted distance between Beijing and Oakland.

The delay of Opt. is the smallest and it always remains the same with the increasing of DCs. Although the delay of BWF is slightly bigger than that of Opt., in average 5%, we find Opt. can only optimize the optimal objective value and cannot optimize other clusters' delay. For example, both NPA-PDP and BWF select DCs 2, 6, 9. Opt. selects 2, 5, 7. For the three algorithms, users 1, 2, 3 are all served by DC2. But Opt. selects DCs 5 and 7 to serve other users, while NPA-PDP and BWF select DCs 6 and 9 to serve other users. Assignments and delay of each user are compared in Table 3. Obviously the overall delay incurred by Opt. is 335296, which is bigger than those of other two algorithms, i.e., 227851. This is due to that Opt. cannot take into consideration of all clusters.

Table 3. Opt. cannot optimize the delay of each cluster

Algorithms	DC	User 4	User 5	DC	User 6	User 7	User 8	User 9	Sum
Opt.	DC 5	25189	23172	DC 7	9496	54108	88790	134541	335296
NPA-PDP, BWF	DC 6	6679	5137	DC 9	17893	18542	38263	141337	227851

Execution time shown in Figure 6 indicates that Opt. (Note that the execution of NPA-PDP almost overlaps that of Random.) needs more time than other algorithms. With the increasing of DCs, the time consumption of Opt. increases sharply, almost at an exponential rate, while BWF increases gently.

We also increase the number of users from 9 to 19 when there are 10 DCs. Every time two users are added in the sequence of user number. The delay is depicted in the Figure 7.

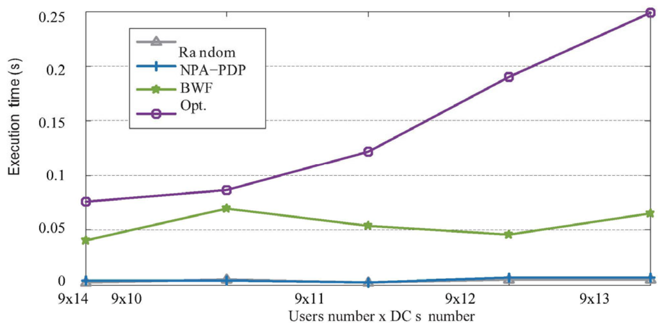


Figure 6. Execution time when the number of DC increases

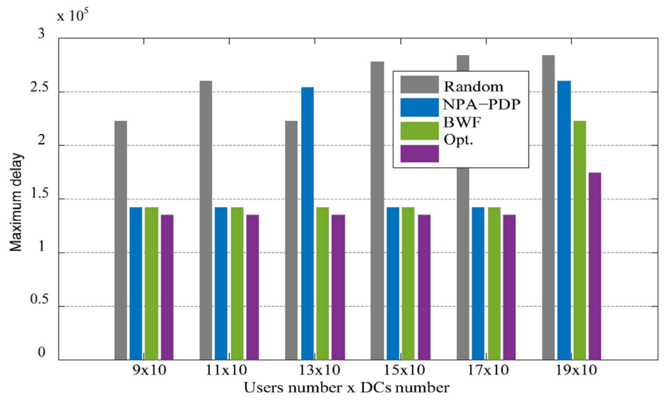


Figure 7. Maximum delay when the number of users increases

The delay changes after user 18 and 19 are added because their smallest weighted distances are bigger than the current delay. This also reveals that it is unnecessary to use more DCs when users are added. Only when farther users are added then it maybe leads to bigger delay. Though Opt. finds the optimum. It is again observed that the overall delay incurred by Opt. is bigger than that of BWF. Limited to the randomness of the selection of user locations, it cannon depict a monotone increasing delay curve. The simulation still demonstrates a tendency that, when the number of users increases the delay will not decrease.

It is noteworthy that when Users 12, 13 are added, the delay of NPA-PDP increases. This is because that NPA-PDP only deletes a certain DC simply without considering the grouping of users. It reflects the instability of NPA-PDP.

When the number of users increases, the execution time of the algorithms demonstrates a similar tendency to that of Figure 6. In a large scale application where there are more DCs and users, Opt. will incur more time. Comparatively, BWF is more suitable for scale application.

5.2.2 Effect of Parameters

We varies k from 2 to 7 to verify its effect on delay in Figure 8. It is demonstrated that Random incurs the biggest delay. Both NPA-PDP and BWF can find the optimum in the scenario of $k=3$ where the objective values are a little bigger than the optimum value (about 5%). But compared with other algorithms which return 7 DCs, when $k = 7$, BWF returns 6 DCs to serve all users without more delay. This reveals that BWF can find the smallest number of DCs to satisfy the request of users.

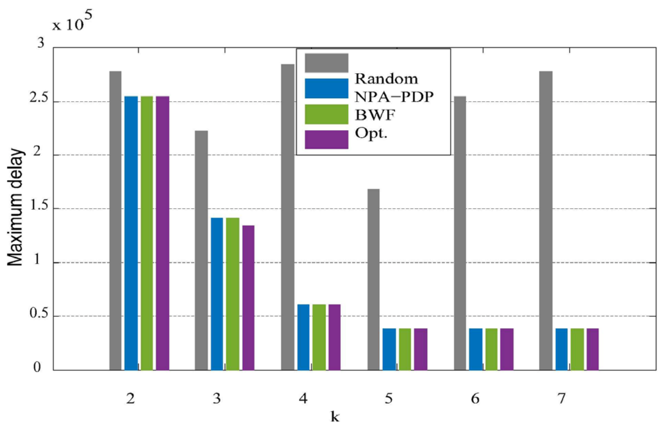


Figure 8. Maximum delay when k increases

It shows a common tendency that, when the number of DCs increases the delay decreases in general. It is because that more candidate DCs permit users to be assigned to nearer DCs. Hence the biggest weighted distance becomes smaller.

When k increases, the execution time of the algorithms is illustrated in Figure 9. This reveals again that BWF is more suitable for scale applications.

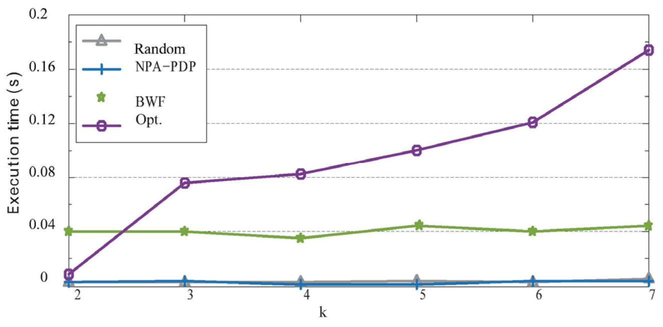


Figure 9. Execution time when k increases

To evaluate whether the algorithms can take into account the weights of the users, we replace the weights of users in the second last line with the weights in the last line in Table 2. The results are listed in Table 4. NPA-PDP, BWF and Opt. all find the optimal solution (the last line). Since in general, the modified weights become smaller, the maximum weighted distance becomes smaller now. The bold DC number indicates the change of DC. After usage weight changing, all algorithms except Random change DC 2 to DC 3. This is because that usage weight of user 2 (WDC) becomes bigger and it is closer to DC 3 (Toronto) than to DC 2 (SFO). Random changes DC 8 to DC 9 because that usage weight of user 6 (BKK) becomes bigger and it is closer to DC 9 (HK) than to DC 8 (Tokyo).

Table 4. Solutions change when usage weights change

	Random	NPA-PDP	BWF	Opt.
w, delay	222559	same as BWF	141337	134540
DC(users)	Omitted	same as BWF	**2**(1,2,3), 6(4,5), 9(6,7,8,9)	**2**(1,2,3), 6(4,5), **8**(6,7,8,9)
$w-M$, delay	110834	same as BWF	54108	same as BWF
DC(users)	Omitted	same as BWF	**3**(1,2,3), 6(4,5), 9(6,7,8,9)	**9**(6,7,8,9)

5.3 Simulation Results for CMDP

To evaluate the performance of the proposed algorithm NPA-CMDP, other three schemes are used for comparison, i.e., (1) Random. (2) Opt.: lpsolve. (3) Clique set based cost minimization (CSCM): it finds clique sets for all threshold graphs first. Then it finds the minimum cost from all the cliques sets which cardinalities are not bigger than k . For each clique, the scheme to find the DC is similar to that of BWF, except that the sum of the cost is compared and the DC with minimum sum is selected as the target DC for this clique. Here we think latency and cost be of the same importance, therefore both α and β are set as 1.

5.3.1 Scalability Evaluation

The number of DCs is also increased from 10 to 14 as in Section 5.2. Figure 10 depicts the performance.

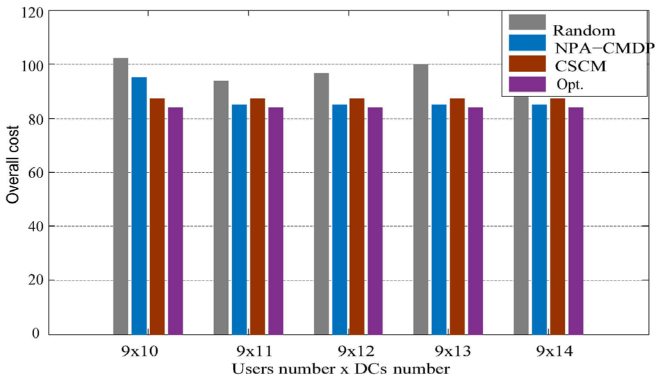


Figure 10. Overall cost when the number of DCs increases

The figure reveals the tendency that the overall cost decreases with the increasing of the number of DCs. Random incurs the biggest cost. Opt. finds the smallest cost. NPA-CMDP and CSCM find medium cost. In general, NPA-CMDP finds smaller cost than CSCM.

When users are increased from 9 to 19 as in Section 5.2, the cost is shown in Figure 11. With users increasing, the overall cost also increases for all algorithms. Opt. finds the smallest cost. NPA-CMDP is better than CSCM. Random is the worst.

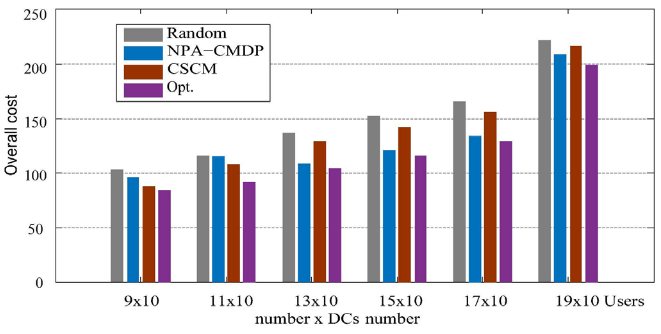


Figure 11. Overall cost when the number of users increases

5.3.2 Effect of Parameters

Figure 12 shows the results when k increases from 2 to 7.

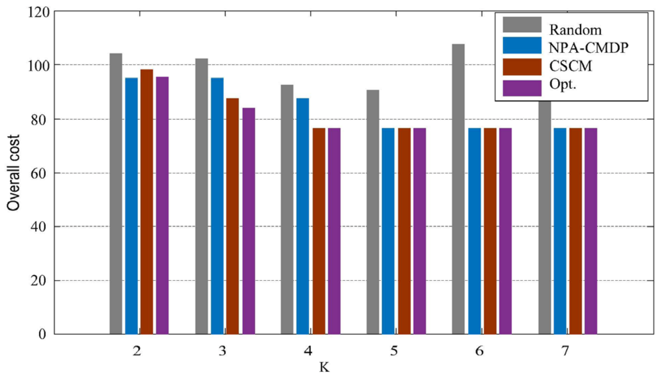


Figure 12. Performance when k increases for CMDP

Although the general tendency is that when the number of target DCs increases, the overall cost is decreasing, after $k \geq 4$, the cost remains unchanged. In reality, both NPA-CMDP and CSCM find the optimal solution.

We first replace usage weight w with $w - M$, and then replace s , c with $s - M$, $c - M$, respectively, to reveal the effect of usage weight and DC weight. The overall cost is detailed in Table 5. We can find that with the weights decreasing, the overall cost also decreases.

Table 5. Solutions change when usage and DC weights change

	Random	NPA-CMDP	CSCM	Opt.
w,c,s	102.0633	95.2356	87.4067	84.03
$w - M$	68.1096	60.1802	57.8713	57.62
$c - M, s - M$	103.3146	81.8068	79.1335	79.13

6 Conclusion

Considering the feasibility of moving geo-distributed big data to cloud, multiple target DCs selection problem is explored in this paper to seek fast access and low cost. The problem generalizes the traditional k-supplier problem, UFL and k-median problem. When the underlying graph is incomplete, a tight 3- approximation algorithm is proposed to accommodate fast access. It can be simplified when the graph is complete. Low cost is achieved by a simple heuristic.

Acknowledgments

This work is financially supported by National Natural Science Foundation of China under Grant No. 11371004 and No. 61672195, National Science and Technology Major Project under Grant No.

2016YFB0800804 and No. 2017YFB0803002, and Shenzhen Science and Technology Plan under Grant No. JCYJ20160318094336513, No. JCYJ20160318094101317 and No. KQCX20150326141251370.

References

[1] D. Agrawal, S. Das, A. El Abbadi, Big Data and Cloud Computing: Current State and Future Opportunities, *14th International Conference on Extending Database Technology*, Uppsala, Sweden, 2011, pp. 530-533.

[2] C.-H. Chi, C. Ding, Q. Liu, Guest Editorial: Knowledge Management and Big Data Analytics, *Journal of Internet Technology*, Vol. 15, No. 6, pp. 937-938, November, 2014.

[3] J. Zhang, H. Huang, X. Wang, Resource Provision Algorithms in Cloud Computing, *Journal of Network & Computer Applications*, Vol. 64, No. C, pp. 23-42, April, 2016.

[4] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, H. Bhogan, Volley: Automated Data Placement for Geo-distributed Cloud Services, *7th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, CA, 2010, pp. 17-32.

[5] Amazon, Amazons3 Global Infrastructure, http://aws.amazon.com/about-aws/global-infrastructure/?nc1=h_ls.

[6] Google, Google Data Centers Locations, http://www.google.com/about/datacenters/inside/locations/index.html.

[7] Amazon, AWS Import/Export, http://aws.amazon.com/importexport/.

[8] P. Yang, Moving an Elephant: Large Scale Hadoop Data Migration at Facebook, https://www.facebook.com/notes/paul-yang/moving-an-elephant-large-scale-hadoop-data-migration-at-facebook/10150246275318920.

[9] Amazon, AWS Direct Connect, http://aws.amazon.com/directconnect/?nc2=hls.

[10] C. Jayalath, P. Eugster, Efficient Geo-distributed Data Processing with Rout, *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, Philadelphia, PA, 2013, pp. 470-480.

[11] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, D. Chen, G-hadoop: Mapreduce Across Distributed Data Centers for Data-intensive Computing, *Future Generation Computer Systems*, Vol. 29, No. 3, pp. 739-750, March, 2013.

[12] C. Jayalath, J. Stephen, P. Eugster, From the Cloud to the Atmosphere: Running Mapreduce Across Data Centers, *IEEE Transactions on Computers*, Vol. 63, No. 1, pp. 74-87, January, 2014.

[13] J. Zhang, L. Zhang, H. Huang, Z. L. Jiang, X. Wang, Key Based Data Analytics Across Data Centers Considering Bi-level Resource Provision in Cloud Computing, *Future Generation Computer Systems*, Vol. 62, No. C, pp. 40-50, September, 2016.

[14] P. Li, S. Guo, S. Yu, W. Zhuang, Cross-cloud Mapreduce for Big Data, *IEEE Transactions on Cloud Computing*, Vol. PP, No. 99, pp. 1-1, August, 2015.

[15] J. Zhao, L. Wang, J. Tao, J. Chen, W. Sun, R. Ranjan, J. Kołodziej, A. Streit, D. Georgakopoulos, A Security Framework in G-hadoop for Big Data Computing Across Distributed Cloud Data Centres, *Journal of Computer and System Sciences*, Vol. 80, No. 5, pp. 994-1007, August, 2014.

[16] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, F. C. M. Lau, Moving Big Data to The Cloud: An Online Cost-minimizing Approach, *IEEE Journal on Selected Areas in Communications*, Vol. 31, No. 12, pp. 2710-2721, December, 2013.

[17] M. Femminella, G. Reali, D. Valocchi, E. Nunzi, The ARES Project: Network Architecture for Delivering and Processing Genomics Data, *2014 IEEE 3rd Symposium on Network Cloud Computing and Applications (NCCA 2014)*, Rome, Italy, 2014, pp. 23-30.

[18] L. Zhang, C. Wu, Z. Li, C. Guo, Moving Big Data to the Cloud, 2*013 IEEE International Conference on Computer Communications*, Turin, Italy, 2013, pp. 405-409

[19] P. Russom, *Big Data Analytics*, The Data Warehousing Institute, 2011.

[20] L. Jiao, J. Lit, W. Du, X. Fu, Multi-objective Data Placement for Multi-cloud Socially Aware Services, *2014 IEEE International Conference on Computer Communications*, Toronto, Canada, 2014, pp. 28-36.

[21] Z. Zhou, F. Liu, R. Zou, J. Liu, H. Xu, H. Jin, Carbon-aware Online Control of Geo-distributed Cloud Services, *IEEE Transactions on Parallel & Distributed Systems*, Vol. 27, No. 9, pp. 2506-2519, September, 2016.

[22] Q. Xia, Z. Xu, W. Liang, A. Zomaya, Collaboration- and Fairness-aware Big Data Management in Distributed Clouds, *IEEE Transactions on Parallel & Distributed Systems*, Vol. 27, No. 7, pp. 1941-1953, July, 2016.

[23] D. S. Hochbaum, D. B. Shmoys, A Unified Approach to Approximation Algorithms for Bottleneck Problems, *Journal of the ACM*, Vol. 33, No. 3, pp. 533-550, July, 1986.

[24] S. Khuller, R. Pless, Y. J. Sussmann, Fault Tolerant K-center Problems, *Theoretical Computer Science*, Vol. 242, No. 1, pp. 237-245, July, 2000.

[25] J. Plesnik, A Heuristic for the P-center Problems in Graphs, *Discrete Applied Mathematics*, Vol. 17, No. 3. pp. 263-268, June, 1987.

[26] J.-H. Lin, J. S. Vitter, E-approximations with Minimum Packing Constraint Violation, *24th ACM Symposium on Theory of Computing*, Columbia, Canada, 1992, pp. 771-782.

[27] S. Guha, S. Khuller, Greedy Strikes Back: Improved Facility Location Algorithms, *Journal of Algorithms-Cognition Informatics and Logic*, Vol. 31, No. 1, pp. 228-248, April, 1999.

[28] N. E. Young, *Greedy Approximation Algorithms for K-medians by Randomized Rounding*, Dartmouth College, 1999.

[29] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit, Local Search Heuristics for K-median and Facility Location Problems, *Siam Journal on Computing*, Vol. 33, No. 3, pp. 544-562, 2004.

[30] Amazon, Amazons3, http://aws.amazon.com/s3/?nc2 =hls.

[31] K. Aardal, P. L. Bodic, Approximation Algorithms for the Transportation Problem with Market Choice and Related Models, *Operations Research Letters*, Vol. 42, No. 8, pp.

549-552, December, 2014.

[32] NSFNET, NSFNET T3 Network, https://en.wikipedia.org/wiki/National Science Foundation Network.

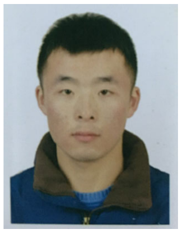
[33] B. Chinoy, H. W. Braun, *The National Science Foundation Network*, SDSC Report GA-A21029, September, 1992.

[34] Sourceforge, lpsolve, http://sourceforge.net/projects/lpsolve/?source=directory.

Biographies

Jiangtao Zhang received the M.S. degree in applied mathematics from Xidian University, China, in 1999. Then as a senior engineer in Huawei, he engaged in research and development of mobile communication networks and cloud computing. He was also in charge of communication network planning and optimization techniques research. In 2016, he received the Ph.D. degree in Computer Science from Harbin Institute of Technology, Shenzhen. He is currently the R&D director of Shenzhen Jingyi Smart Technology Co., Ltd. His research interests lie in the fields of mathematical programming, cloud computing, distributed computing and big data, especially architecture, protocols and algorithms.

Qiang Yuan received the B.Sc (2014) degree in Harbin Institute Of Technology at WeiHai. He is currently working toward the M.Eng degree at the Department of Computer Science, Harbin Institute Of Technology Shenzhen Graduate School, China. His current research interests is cloud computing.



Shi Chen has been pursuing the M.S. degree of computer science and technology in Harbin Institute of Science and Technology, Shenzhen graduate school since 2014. His research interests lies in the fields of cloud computing, green scheduling in cloud data center and spammer detecting in e-commerce.

Hejiao Huang graduated from the City University of Hong Kong and received the Ph.D. degree in computer science in 2004. She is currently a professor in Harbin Institute of Technology Shenzhen Graduate School, China, and previously was an invited professor at INRIA, France. Her research interests include cloud computing, trustworthy computing, formal methods for system design and wireless networks.

Xuan Wang received M.S. and Ph.D. degrees in Computer Sciences from the Harbin Institute of Technology, Harbin, China, in 1994 and 1997, respectively. He is a Professor and Dean of the school of Computer Science and Technology in Harbin Institute of Technology, Shenzhen Graduate School, ShenZhen, China. His research interests include Artificial Intelligence, Computer Network Security, Computational Linguistics, and Computer Vision.