Latency Minimization Through Optimal Data Placement in Fog Networks

Ning Wang and Jie Wu
Department of Computer Science, Rowan University, Glassboro, USA
Center for Networked Computing,Temple University, Philadelphia, USA
Email: wangn@rowan.edu and jiewu@temple.edu

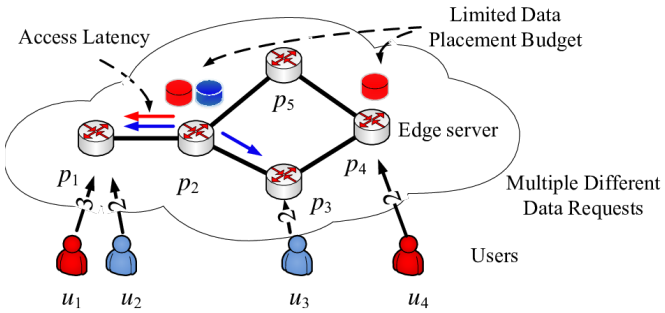
Nowadays, Fog Networks (FNs) distribute services to fog servers so that they are spatially closer to end-users and thus provide high availability and low data access delay, i.e., better usage experience. Given the different data demands of users and limited storage capacity of fog servers in FNs, it is non-trivial to optimally store data. Specifically, in this paper, we discuss the Multiple Data placement with Budget Problem (MDBP), whose objective is to minimize the overall data access latency for all data requests within a given total budget. We start with a case in which data replication is not considered. In this case, we propose a min-cost flow transformation for the MDBP to calculate the optimal data placement strategy. We further propose an efficient local information collection scheme to reduce the time complexity in the tree topology. In a general case with data replication, we prove that the MDBP is non-deterministic polynomial-time (NP)-complete. In the line topology, we can use dynamic programming to solve the single data request scenario. We also apply a novel rounding algorithm which incurs an approximation ratio of 10 in terms of the overall latency in the general case. We validate the proposed algorithms by using the PlanetLab trace, proving that they significantly improve the performance.

Keywords: Fog computing, data placement, and planning.

1 Introduction

With the wide availability of smart devices, people are able to access social websites, videos, and software from Internet anywhere at any time. According to [1], in 2016, Netflix had more than 65 million users, with a total of 100 million hours spent streaming movies and TV shows daily. This accounted for 32.25% of the total downstream traffic during peak periods in North America. Facebook, a famous social media website, had 2.19 billion monthly active users up to the first quarter of 2018 [2]. The traditional mechanism is to store content into the central server, which leads to a relatively long data retrieving latency. To alleviate this issue, Fog Networks (FNs) are proposed and applied in commercial companies. According to Cisco's White Paper [3], global streaming traffic is expected to account for 32% of all consumer traffic on the Internet by 2021, and more than 70% percent of this traffic will cross FNs up, an increase from 56% in 2016.

To solve the problem of the long content retrieving latency when storing data in the center, FNs can geographically store partial or entire data at fog servers deployed close to clients. This will then reduce network and center server loading times and provide a better quality of service,



i.e., low latency to end-clients [4, 5]. Due to the relatively large storage space and bandwidth

Figure 1: An illustration of a typical fog network.

needs of retrieving data, a FN typically employs a limited number of fog servers, each of which has medium or low storage capacity. Such servers can collectively store a larger number of objects and serve clients from all networks with larger aggregate bandwidth needs.

An illustration of the FN network model is shown in Fig. 1. There are 5 fog server nodes where data can be stored to support 2 different data requests, represented by red and blue colors. Note that there is a capacity limit for each server node. In a given time period, we can assume that there are different sets of data requests that arrive based on the prediction. If the data request cannot be satisfied in the local server node, it will search the nearby servers in FNs to find the requested data and there is a corresponding access latency. For example, user u_2 needs to retrieve data from the server p_2, which incurs a transmission from p_2 to p_1. The access latency between a pair of nodes is assumed to be constant over a relatively long period. In addition, there may be multiple identical data requests, for instance, trending news. In Fig. 1, users u_1 and u_4 request the same data and users u_2 and u_3 request another data. The bandwidth and server processing speed are assumed to be sufficient, and thus, multiple identical data requests can retrieve data from the same server node.

In this paper, we address the latency minimization problem in FNs through data placement optimization with limited total placement budget, called the Multiple Data placement with Budget Problem (MDBP). It is motivated by the fact that the data deployment cost is the major cost in FNs [6]. However, MDBP is non-trivial. For example, in Fig. 1, only servers p_2, p_3, and p_4 have available storage and the available slots are 2, 1, and 1. The total budget is 3 and all link latencies are 1. The first challenge is due to the network topology. For example, it is easy to see that p_5 is a bad placement location since it is far from all users. However, p_2 is a good location for user u_1, but bad for user u_4. Therefore, for each data placement, we need to jointly consider its influence to all corresponding requests. The competition between two data requests further complicates this challenge. The second challenge is due to limited budget and request demands. In this example, users u_1 and u_4 have relatively larger demands; these high demand users should have low access latency but it does not mean that we need to always assign them more data copies. For example, it may be the case all the high demand requests are close to each other. The third challenge is due to copy distribution. If we have multiple data for a request, its decision should be further revised. For example, if we use two data copies for data request 1, we can place data at locations p_2 and p_4 to satisfy users u_1 and u_4 so that they both have low access latency. In addition, the limited storage size of each server location further complicates the proposed problem. It is because we cannot simply apply the individual solutions of each singe data requests since they may violate the server storage constraint.

We discuss the solution for the MDBP in two different cases depending on whether there is data replication or not. In the first case, we do not consider data replication so that we can focus on optimizing data placement. We find that we can transfer the MDBP into a min-cost flow formulation and thus find the optimal solution. We also propose a local information collection method to reduce the time complexity of the min-cost flow transformation in the tree topology, the common topology of FNs. In a general case with data replication, the MDBP turns out to be NP-complete even when there is only one type of data request. We propose a dynamic programming solution to find the optimal solution in the single request scenario with line topology. In the general case, we first propose a heuristic algorithm. However, its performance is related to the system environments. In addition, we further propose a novel assignment approach based on the rounding technique. It first finds the optimal fractional solution and then gradually transfers into the integer solution without avoiding the server capacity constraint. The proposed approach is proved to have an approximation ratio of 10.

The contributions of this paper are summarized as follows:

- To our best knowledge, we are the first to consider the optimal data placement with budget in FNs for multiple data requests under the server capacity constraint.
- We find the optimal solution of MDBP in the scenario where there is no data replication through min-cost flow formulation, and propose a local information collection method to reduce the time complexity.
- In the general case, we prove that the MDBP is NP-complete. A novel rounding approach is proposed, and it has a constant approximation ratio of 10.
- We verify the effectiveness of the proposed approaches using the PlanetLab trace, which is a worldwide Internet trace.

The remainder of the paper is organized as follows: The related works are in Section II. The problem statement is in Section III. The min-cost flow formulation is provided in Section IV for the data placement without replication. In the general scenario, we first prove that the proposed problem is NP-complete and then propose a novel rounding solution with approximation analysis in Section V. The experimental results from the PlanetLab trace are shown in Section VII, and we conclude the paper in Section VIII.

2 Related Work

The data placement problem [7, 8] in FNs is a fundamental problem. The existing work can be briefly categorized into two types based on whether the data placement changes with time.

2.1 Long-Term and Short-Term Placement

Early work in [9] studied the k-cache placement problem in a given network with special topologies, i.e., line and ring. The difference between [9] and our paper is that in [9], each data can fractionally support clients, but this assumption is not true in this paper. The work in [10] studied the trade-off between selecting a better traffic-delivery path and increasing the number of FN servers. In recent work in [6], the authors considered the dynamic network demand and found that deploying data close to the end-users might not always be the optimal solution. In [1], to find the optimal on-demand content delivery in short-term (e.g., one-week), the authors formulated and solved the content placement problem with the constraints of storage space and edge bandwidth. In [11], the authors discussed the optimal data update frequency decision in data placement. In [12], the authors provided a hierarchical data management architecture to maximize the traffic volume served from data and minimize the bandwidth cost.

Table 1: Summary of symbols

Symbol	Interpretation
m	total type of data request
n	number of fog servers
p_i	server i
s_i	server p_i's storage size
l_{ij}	access latency to between servers p_i to p_j
d_{ij}	demand of the data request j at server i
x_{ij}	server node i has data j
θ	data placement budget
$\mathbb{X}$	overall data placement planning
$c(d_{ij}, \mathbb{X})$	latency of d_{ij} with a placement planning $\mathbb{X}$

2.2 Data replication

The work in [13] addressed the facility location problem with different client types, which is equivalent to the case of data placement without replication since there is only one facility for a type of client. Authors in [14] discuss the optimal single data request placement with replication in the tree topology. The optimal solution is obtained through the dynamic programming technique. In [15], data can be transferred between a set of proxy servers and thus the authors optimized the data transformation cost. In [16], authors jointly considered optimization of data replication costs and access latency. In [17], they jointly consider the data receiving for multiple different data. Therefore, the co-locations of data are important.

This paper addresses long-term data placement with data replication. Our work differs from existing works by considering multiple data requests with different demands and limited total placement budgets.

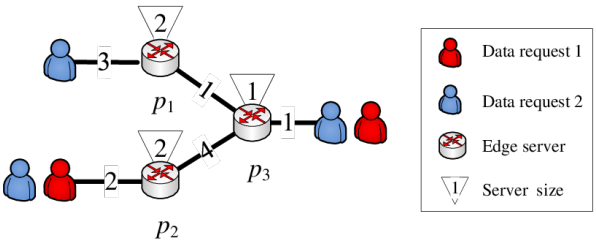
3 Problem Statement

In this section, we discuss the proposed network models, followed by the problem formulation and challenges.

3.1 Network Model

A Fog Network (FN) is an overlay network over the Internet; it is composed of a set of fog servers. The server nodes are potential places for data deployment. Without loss of generality, we model the topology of the overlay network with a connected undirected graph $G(V, E)$, where V is the set of vertices denoting the servers, and E is the set of edges denoting the data transmission between servers in the network and the edge weight is the corresponding latency. We assume that the topology of the target network is known in advance and there is a total of n server nodes in the FN, i.e., $|V| = n$. Note that there is a storage limit in each server, denoted by s_i.

This paper considers an offline scenario. We assume that there are m different data in total. For each data, there can be one or multiple data requests (up to n). A data request is denoted by d_{ij}, which means there is a request from location i for the data j and the value of d_{ij} is the total demand at that location. Let $\mathbb{X} = \{x_{11}, x_{12}, \cdots, x_{nm}\}$ be a feasible placement plan. For



Strategy	Distribution	Placement	Latency Sum
1	$(1d_1, 3d_2)$	p_1:{d_1,d_2}, p_2:{d_2}, p_3:{d_2}	15
2	$(1d_1, 3d_2)$	p_1:{d_2}, p_2:{d_1,d_2}, p_3:{d_2}	13
3	$(2d_1, 2d_2)$	p_1:{d_1}, p_2:{d_1,d_2}, p_3:{d_2}	11
4	$(2d_1, 2d_2)$	p_1:{d_2}, p_2:{d_1,d_2}, p_3:{d_1}	10

Figure 2: An illustration of the problem's challenges.

each data request, it will be satisfied by the nearest server that stores the data in the FN. The nearest data can be found through the shortest-path algorithms. Therefore, for a data request d_{ij}, its latency for a given placement planing $\mathbb{X}$ is $c(d_{ij}, \mathbb{X})$, which is

$$c(d_{ij}, \mathbb{X}) = d_{ij} \times \min_{i' \in [1,n] \& x_{i'j}=1} l_{ii'}. \quad (1)$$

Currently, we assume that all data requests are the same size. In the future, we might extend it to a more general case where different data requests might have different sizes.

3.2 Multiple Data Placement with Budget Problem

To meet all data requests and minimize data access latency, the system provider can place data in the server nodes of FNs. However, the data placement can be costly. Therefore, we propose the Multiple Data placement with Budget Problem (MDBP) to minimize the average data access latency for the FN. Specifically, the MDBP is as follows - we would like to plan the data placement locations to minimize the user's data access latency while avoiding the server's capacity. Based on whether there is replication or not, we consider two versions: (1) There is only one data for each data request in the network. (2) There may be multiple data for a data request, but there is an overall data placement budget. The problem can be mathematically formulated as follows.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c(d_{ij}, \mathbb{X}) \quad (2)$$

$$\text{s.t.} \sum_{j=1}^{m} x_{ij} \leq s_i \quad \forall i, \quad (3)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \leq \theta \quad \forall i, j, \quad (4)$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j, \quad (5)$$

where $x_{ij} \in \mathbb{X}$ is a placement decision. The first constraint, i.e., Eq. 3, means that each server cannot exceed its capacity. The second constraint, i.e., Eq. 4, is the total budget constraint where θ is the given placement budget. The last constraint, i.e., Eq. 5, means that data cannot be partitioned.

Note that in the aforementioned formulation, the MDBP does not consider the users' mobility [17], that is, a data request represents a user. However, it can be easily extended to a case that considers users' mobility. The idea of the conversion is that we can transfer a mobile user into multiple static users in the MDBP, where the demand of each static user is the percentage of staying at that mobile node. An example is shown in Fig. 1, where there are 2 users. User 1 spends

60% of its time in p_1, and 40% of its time in p_4. User 2 spends 50% of its time in p_1 and p_3. After the conversion, we can apply the solution of MDBP to the extended model.

3.3 Challenges

The proposed MDBP is non-trivial even when the user request pattern is given/predicted. It is because we need to jointly consider the data placement distribution for multiple data requests. We cannot simply consider each data request individually and then combine the solutions. The reason is that it might be the case that there are optimal locations for multiple data requests, but the server storage size is not large enough to hold all of them. Given the data replication, how to jointly distribute the data in each type of request another challenge. An illustration of the challenges of the MDBP is shown in Fig. 2, where there are two different types of data requests and three available servers. We ignore the access servers in this example. The weights on the edges represent the corresponding communication latency. In this toy example, the storage size of each of these three servers is 2, 2, and 1, respectively, and the total data budget in the network is 4. In this example, all data requests have the unit demand to simplify the illustration. We propose four different strategies to minimize the overall latency and show the placement challenge. First, if we place one data copy for data request 1 and three data copies for data request 2, we find that improper placement leads to large latency. In strategy 1, the latency for data request 1 is $7 + 2$, and the latency for data request 2 is $3 + 2 + 1$. Therefore, the overall latency is 15. In this toy example, strategies 1 and 2 and strategies 3 and 4 have the same data distribution but different latencies. In addition, this example shows that different data distributions influence on the latency, i.e., strategies 3 and 4 have lower latency compared to strategies 1 and 2.

4 Delay Minimization Without Replication

In this section, we focus on the storage placement optimization without replication. The problem formulation is introduced first, followed by the min-cost flow solution.

4.1 Problem Formulation

The MDBP can be simplified since different data have no influence on each other in terms of the latency and thus, Eq. 1 can be re-formulated as follows.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{n} l_{ik} d_{kj} x_{ij} \tag{6}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j, \tag{7}$$

$$\sum_{j=1}^{m} x_{ij} \leq s_i \quad \forall i, \tag{8}$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j, \tag{9}$$

where x_{ij} is a decision variable to show that the server p_i has a data j. The first constraint is that each data can be placed at one location and only one location in the network and therefore, there is no replication. The second constraint is that the total amount of data placed at a server node cannot exceed its capacity constraint.

4.2 Min-Cost Flow Formulation

In this subsection, we would like to explain that the MDBP can be solved by the min-cost flow formulation.

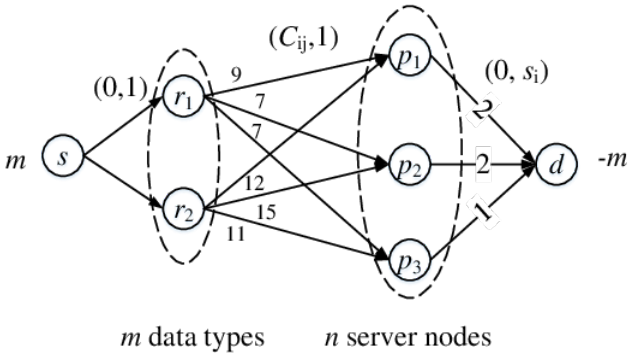


Figure 3: An illustration of the min-cost transformation.

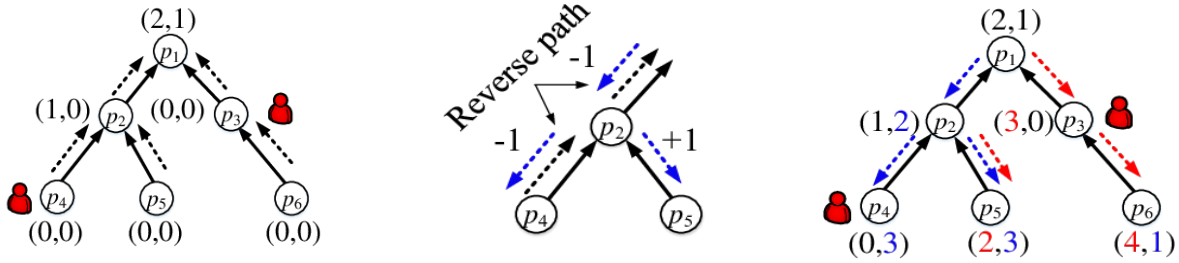
Theorem 1 *The MDBP problem is equivalent to the min-cost flow problem.*

Proof. We prove it by showing the step-by-step transformation. The solution of the MDBP can be considered as a matching process. For each type of data, we generate a demand node, and thus, there are m demand nodes in total, i.e., $\{r_1, r_2, \cdots, r_m\}$. Similarly, we generate a node for each server node and thus, there are n server nodes $\{p_1, p_2, \cdots, p_n\}$. For each demand node r_i, there are n edges for all server nodes. The corresponding weight of a node represents a possible placement decision to place data into that server in the MDBP. The unit edge cost, C_{ij} is the corresponding latency summation to store the data in that location for all corresponding types of data request(s), $C_{ij} = \sum_{i=1}^{n} \sum_{k=1}^{n} l_{ik} d_{kj}$. The edge capacity is 1. In addition, there is a virtual source and a virtual destination. Each demand node has an edge with the virtual source, where the corresponding unit edge cost is 0 and the edge capability is 1. Each server node has an edge with the virtual destination, where the corresponding unit edge cost is 0 and the edge capability is s_i. The demand of the min-cost flow is set as the total number of data request, i.e., m, in the network and the demand is generated at the virtual source. All demands are consumed at the virtual destination. According to the max-flow theory, if all edges' capacities are integral, the final solution is always an integer solution based on the augmenting path method [19]. Then, if there is a flow in an edge which connects the data request and server location, we can use the same data placement between the data node and the server node in the MDBP. Since this assignment leads to the same latency cost as that in the corresponding min-cost flow problem. The total demand of the virtual sink ensures that each data type has to be matched in the MDBP. The capacity of links between the server node and the destination node ensures that each server cannot exceed its corresponding storage capacity in the MDBP.

Fig. 3 shows the min-cost flow formulation for the example in Fig. 2. Each data request can be assigned to any server. The capacity constraint of each server location is controlled by the edge flow constraint between the server node and virtual destination in the min-cost flow formulation. For example, the cumulated latency for data request 1 in three servers is 9, 7, and 7, respectively. The min-cost flow can be solved by the successive shortest path [20].

4.3 Complexity Reduction

In this subsection, we design an efficient local information collecting method which can reduce the overall time complexity of cumulated latency calculation in min-cost flow calculation. The time complexity of calculating the cumulated latency of each type of data request in all server locations is $O(n^2)$ in the simple approach since we need to traverse the network for each data



(a) demand collection (b) updating rule (c) cemand updating

Figure 4: An illustration of latency updating procedure.

request. Here, we show that the overall time complexity can be reduced to $O(n)$ in the tree topology, which is the common topology in FNs [21].

Theorem 2 *The cumulated latency calculation of a data request can be finished in* $O(n)$ *in the tree topology*.

Proof. We prove this theorem by introducing the $O(n)$ calculating method.

The procedure is as follows: We transfer G into a tree by considering an arbitrary node as the root node. Then, we aggregate the latency cost gradually from the leaves to the root of the networks. For each node i, it keeps a vector of $C_i, C_i = \{c_1, c_2, \cdots, c_k\}$, where k is the total number of data requests. It records its distance to all the data request access locations. In Fig. 4, $k = 2$. Initially, all elements in the vector are 0. Then, we calculate the total latency of the data placement in any node by using the following two steps:

• *Upward information collection:* we gradually update the latency value of each element c_j in C_i from leaf nodes to the root node as follows. If we have seen the corresponding data request from a successor node i', then $c_j = c_j + l_{ii'}$. Otherwise, c_j keeps its original value.

• *Downward information updating:* we gradually update the latency value in the revised visiting order to summarize the information from different branches. If the branch is used in the information collection and the corresponding data is seen again, then $c_j = c_j - l_{ii'}$. Otherwise, $c_j = c_j + l_{ii'}$.

After the information updating is done, the placement cost of a location is $\sum_{i=1}^{k} c_i$. Since the each node will be only visited twice, the overall time complexity is $O(n)$.

An illustration is shown in Fig. 4, where all edge weights are unit. Since the calculation is independent for different data requests, we use a data request to illustrate the procedure. In Fig. 4, the data requests generate at server nodes p_3 and p_4. There are two data requests in this example. Eeach server node keeps a bracket, where each element records the distance from the current location to the corresponding data request location. Note that in the upward information collection procedure, each node only collects the distance information of the data requests in its sub-tree. Therefore, at the end, only the root node has the information of all data requests. In downward information updating, the information collected by the root is distributed in all branches. The data updating of the node p_2 illustrates one case, where the corresponding distance decreases by 1. The data updating of the node p_4 illustrates two cases: (1) 3 increases by 1 for data request

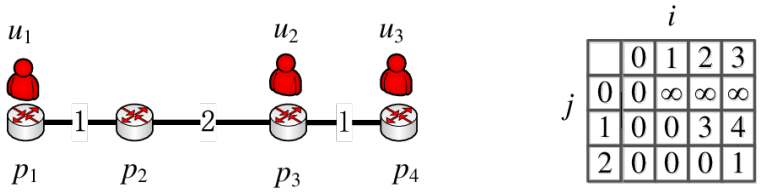


Figure 5: The dynamic programming in the line topology.

1, and (2) 0 increases by 1 for data request 2, since the distance to data requests 1 and 2 increases. Note that here, we use a binary tree as an example, but this method works in any tree topology.

5 Delay Minimization With Replication

In this section, the problem hardness in the general case is discussed first, followed by the optimal solution in a line topology, and the solution in the general scenario.

5.1 Hardness Proof

Theorem 3 *The proposed MDBP is NP-complete.*

Proof. We prove the proposed MDBP is NP-complete by first proving that it belongs to the NP-class. For a given placement, we can verify whether all constraints are satisfied simultaneously in polynomial time. Now, we show its NP-completeness by a reduction of the k-median problem [22].

The k-median problem is as follows: in a metric space, $G(E,V)$, there is a set of clients $C \in V$, and a set of facilities $F \in X$. We would like to open k facilities in F, and then assign each client node j to the selected center that is closest to it. If location j is assigned to a center i, we incur a cost c_{ij}. The goal is to select k centers so as to minimize the sum of the assignment cost.

The reduction from a special case of the MDBP to the k-median problem is as follows: in a special case of the MDBP, we assume that there is only one type of data request. In this setting, there is no capacity constraint since each available server node should have at least one storage space. Assume that the total budget is k. We need to determine the data placement location, which is equivalent to finding the centers (facilities) in the k-median problem. Its weight is the total latency cost for the data request at the corresponding server location. Clearly, if we set the latency as the corresponding c_{ij} in the k-median problem, we can apply the solution of the MDBP to the k-median problem.

5.2 Single Request in Line Topology

We have proven that the MDBP problem is NP-complete in the general graph even in the simple request case. However, when the FN has some particular topology, i.e., the line topology, we can determine the optimal placement for that user by using dynamic programming in the single data request scenario. Specifically, we first sort all data requests directionally. Then, we can define a state called $opt[i,j]$ which represents the minimum cost for that data request up to the first i requests with j copies. Then, we can update $opt[i,j]$ as follows.

$$opt[i,j] = \min \begin{cases} opt[i,j-1] \\ opt[i',j-1] + \min\limits_{p_k \in [p_{i'+1}, p_i]} c_k[i'+1,i], \forall i' < i, \end{cases} \quad (10)$$

Algorithm 1 Dynamic Programming for Single Request

Input: Job information and rental cost function

Output: The placement result $\mathbb{X}$.

1: Initialize $opt[i, j] = \infty$ except $opt[0, 0] = 0$.
2: **for** Each request i from 0 to m **do**
3: **for** Each request i' from 0 to i **do**
4: **for** Each server location between i' and i **do**
5: Update $opt[i, j]$ based on Eq. 10.
6: **return** Return the assignment of $opt[n, \theta]$.

where i' is a request prior to request i. The $c_k[i' + 1, i]$ is the latency for assigning a new data copy at server p_k, which is between $p_{i'}$ and p_i to cover data requests in this range. The idea behind Eq. 10 is that the optimal solution always falls into one of two cases: (1) we have moved to the location i without adding one more data copy to get the optimal result; (2) we have moved to the location i and we can use one more data copy to reduce the overall latency.

A toy example can be used to illustrate the proposed dynamic programming in Fig. 5. In this example, let us assume that $\theta = 2$, which means that we can use 2 data copies at most. Initially, we can use only one data to cover three users. After calculating the four available server locations, $opt[1,1] = 0, opt[1,2] = 3$, and $opt[1,3] = 4$, which achieve the optimal value when the data is put at p_1, p_2, and p_3. Then, we add one more data copy into the network. It is easy to calculate $opt[2,1] = 0, opt[2,2] = 0, opt[2,3] = 1$. Here is an example of a calculation of $opt[2,3]$.

$$\begin{aligned} opt[i,j] \quad &= \min\{opt[0,0] + c_3[1,3], opt[1,1] + c_3[2,3], \\ &\quad\ opt[2,1] + c_4[3,3]\}, \end{aligned} \tag{11}$$

In Eq. 11, we use the $\min_{p\in[i'+1,i]} c_p[i' + 1, i]$, and ignore the calculation procedure. It is clear that $opt[1,1] + c_3[2,3] = 1$ is the minimum in this example.

5.3 Greedy Solution in Multiple Requests

If there are multiple different requests, we cannot simply apply the optimal solution for each request because it might not make a feasible solution if we add them together.

To address this problem, we first propose the following heuristic algorithm as shown in Algorithm 7. Initially, when the data request is not covered, we go through all types of data requests and calculate their optimal data placements so far. The data whose placement leads to the minimum latency in each round will be selected to put into the network. It is shown in lines 1 to 4. After that, there is one data for any data request in the network to ensure the coverage constraint. Then, for each round, we pick the data, which can reduce the latency maximum if that data can change its location once, shown in lines 5 to 8. However, the proposed heuristic may be far from optimal. An example is shown in Fig. 6, where there is one slot fog server location 1 and 3 to store data. The greedy solution is shown in Fig. 5. It will select the blue data request in the first round due to the smallest latency increase. However, this option leads to the large latency at the second round. The optimal solution is shown in Fig. 5, where the placement decision jointly considers two data requests.

(a) greedy (b) optimal

Figure 6: An illustration of the greedy algorithm.

Theorem 4 *The proposed heuristic algorithm does not have an approximation ratio smaller than* 3ρ, *where* ρ *is the maximum data request ratio in the network over the time.*

Proof. We can prove Theorem 4 through an extreme example. We propose a contradiction example in the line topology, where each server location has the unit storage capacity. Assume that there is a set of data requests at locations $\{p_1, p_2, p_3, \cdots, p_k\}$, and for each data request, there is only one location. Therefore, we use $\{p_1, p_2, p_3, \cdots, p_k\}$ to denote the data request locations in this proof. In addition, we assume that there exist server locations in $\{p_0, (p_1 + p_2)/2, (p_2 + p_3)/2, \cdots, (p_k + p_{k+1})/2\}$ where $p_0/2$, and $p_{p+1}/2$ are locations to the left of p_1, and right of location p_k, respectively. In addition, $p_1 - p_0 < (p_2 - p_1)/2$, and $p_{k+1} - p_k > (p_k - p_{k-1})/2$. Therefore, according to the heuristic algorithm, we will put all the data into the first server at the right except the last one. The overall latency is

$$\sum_{i=1}^{k} (p_{i+1} - p_i)/2 + p_k - p_0 = (p_{k+1} - p_1 - p_0)/2 + p_k \quad (12)$$

Instead, if we assign each data request to the first server location at the left, the overall latency is

$$\sum_{i=1}^{k+1} (p_i - p_{i-1})/2 = (p_{k+1} - p_0)/2 \quad (13)$$

Then, $\frac{p_{k+1}-p_0}{2} < \frac{p_{k+1}-p_1-p_0}{2} + p_k \approx 3\frac{p_{k+1}-p_0}{2}$ when k is a large number. The number of data requests at a time also has an influence on the ratio and the ρ, which is the maximum number of requests in the network over time. Therefore, the overall approximation ratio is 3ρ.

According to Theorem 4, we know that the proposed heuristic algorithm cannot achieve good performance even in the line topology. Therefore, it is necessary to propose an approach which can guarantee adequate performance in different network environments.

5.4 Rounding Approach in Multiple Requests

To improve the performance of the greedy solution, we propose a two-step rounding solution. In the first step, we relax the proposed problem into the Linear Programming (LP) and obtain the lower bound of the MDBP. Then we propose a novel rounding technique, which first rounds a half-integral solution through the min-cost flow. Then, we can further convert the half-integral solution to an integer solution.

5.4.1 Generating Linear Programming Solution

The linear programming formulation of the MDBP problem is,

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{h} l_{ij} d_k y_{ijk} \quad (14)$$

$$\text{s.t.} \sum_{i=1}^{n} y_{ijk} \geq 1, \forall j, \quad (15)$$

$$y_{ijk} \leq z_{ij}, \quad \forall i, j \quad (16)$$

$$\sum_{j=1}^{m} z_{ij} \leq s_i, \quad \forall i, \quad (17)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} z_{ij} \leq \theta, \quad \forall i, \quad (18)$$

$$y_{ijk}, z_{ij} \in [0,1], \quad \forall i, j \quad (19)$$

Algorithm 2 Min-Volume Algorithm

Input: Data request distribution and network configuration

Output: The placement result $\mathbb{X}$.

while uncovered data request **do**
  for data requst i from 1 to m **do**
    for server location j from 1 to n **do**
      Check the placement cost of x_{ij}.
  $x_{ij} = \arg\min_{x} c(d_{ij}, \mathbb{X}) - c(d_{ij}, \mathbb{X} \cup x_{ij}) - \mathbb{X})$
while The movement budget θ is not reached **do**
  for data requst i from 1 to m **do**
    for server location j from 1 to n **do**
      Check the placement cost of x_{ij}.
  $x_{ij} = \arg\min_{x} c(d_{ij}, \mathbb{X}) - c(d_{ij}, \mathbb{X} \cup x_{ij}) - \mathbb{X})$
Return the placement result.

Algorithm 3 Rounding Algorithm

Input: Data request distribution and network configuration

Output: The placement result $\mathbb{X}$.

1: Calculate the linear programming solution of MDBP.
2: Aggregate demand into centers based on the geo-distance.
3: Convert the fractional solution to the half integer solution.
4: Convert the half integer solution to the integer solution.
5: Return the placement result.

which is formulated from the angle of each data request. Therefore d_k is the demand for a data request k. Let us assume that the total number of data requests is h. Note that $h \geq m$ due to the repeated data requests. In this formulation, y_{ijk} means the server node i has data j and covers data request k fractionally, z_{ij} means that server i has z_{ij} amount of data j. Eqs. 15 and 16 ensure that each data request has to be satisfied and the assignment is feasible. Eq. 17 is the capacity constraint for each server, and Eq. 18 is the total placement budget constraint.

5.4.2 Creating centers

Since it is hard to directly convert the fractional solution to integral solution, we would like to aggregate the assignment into several "center" servers, so that each center has at least a half data. To simplify the following explanation, we create the notation $L_k, L_k = \sum_{i=1}^{n} l_{ij} y_{ijk}$, which is the unit demand cost of the LP solution for data request k. Let us consider all the data requests that need data j. We sort them in increasing order of the C_k. Then, for each data request k, if

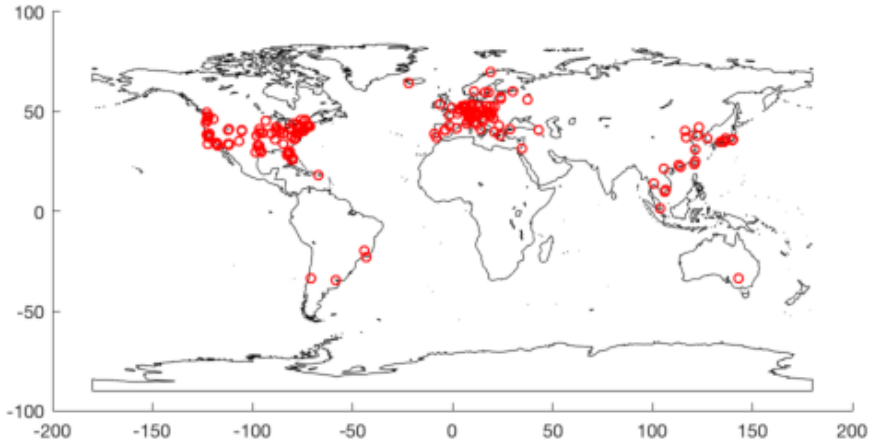


Figure 7: Server distribution.

there is an another data request k' and $l_{kk'} < 4\max(L_{k'}, L_k) = 4L_k$, we would like to consolidate the demand on data request d_k to d'_k, that is, $d'_k = d'_k + d_k$. We apply this procedure to all data requests; the remaining servers with non-zero demands are called center servers. Since each data moves at most $4L_k$, it is clear that any solution can incur an additive factor of at most $4OPT$.

5.4.3 Converting to integral solution

After we get the data center servers, an important issue is that how we can assign data requests so that the result is an integral solution that doesn't violate the server's capacity constraint. We refer to [22] to propose a two-phase solution where the first step is to build a half-integral solution. This ensures that the distance between a data request and the server serving it is fractionally bounded by the access cost. Therefore, the fractional solution is equivalent to a feasible flow to a min-cost flow problem with integral capacities. Note that to apply the solution in [23] to the MDBP, we need to add a virtual node before going to the destination with the link capacity as the total budget. With the property of the min-cost flow, we can always find an integer solution of no greater cost. By applying the min-cost flow transformation in [23], it has an approximation ratio of 6.

The proposed algorithm has a constant approximation ratio of 10. The center creation incurs at most $4OPT$. The half-integral solution transformation introduces at most $3OPT$. The integer solution conversion further leads to $2OPT$. Therefore, the overall cost is $4 + 2 \times 3 = 10OPT$.

6 Performance Evaluation

We evaluate the performance of the proposed solution in this section. The compared algorithms, the trace, the simulation settings, and the evaluation results are presented as follows.

6.1 Trace Information

In this paper, we use the PlanetLab trace [24] generated from the PlanetLab testbed. PlanetLab is a global research network that supports the development of new network services. It contains a set of geo-distributed hosts runing PlanetLab software worldwide. In this trace, the medians of all latencies, i.e., RTT, between nodes are measured through the King method. 325 PlantLab nodes and 400 other popular websites are measured.

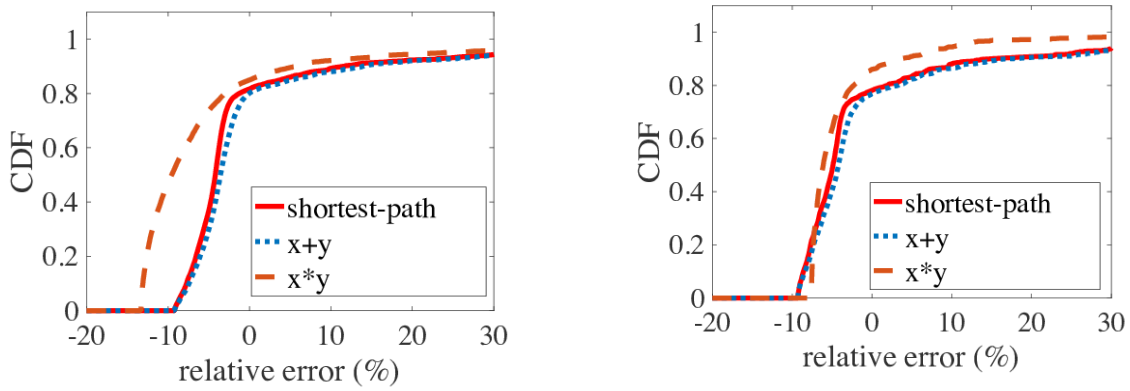


Figure 8: Latency-distance mapping.

In the PlanetLab trace, the domain of each node is provided. Each node's geometric location is retrieved through the domain-to-IP database and the IP-to-Geo database, provided by [25] and [26], respectively. Some domains are no longer in service. In the end, there are 689 nodes. It is reported that the mapping error is within 5 mi and can be ignored in our experiments. Fig. 9 shows the trace visualization results.

6.2 Experimental Setting

We conduct experiments on two scales, i.e., the world and the United States. At the United States scale, we use the nodes on the west coast to simulate the line-topology. The number of data requests, m, changes from 2 to 5. The number of users changes from 10 to 20, which are randomly selected from the first 325 nodes in the PlanetLab trace. The data request location is randomly generated in each round. The pair latency is known and therefore, the topology is not important in the experiments. The number of data placement budgets also changes in the experiments from m to $2m$. We change the following four settings in the experiments: (1) the number of data budgets, (2) the number of data, (3) the number of the data requests, and (4) different server capacities.

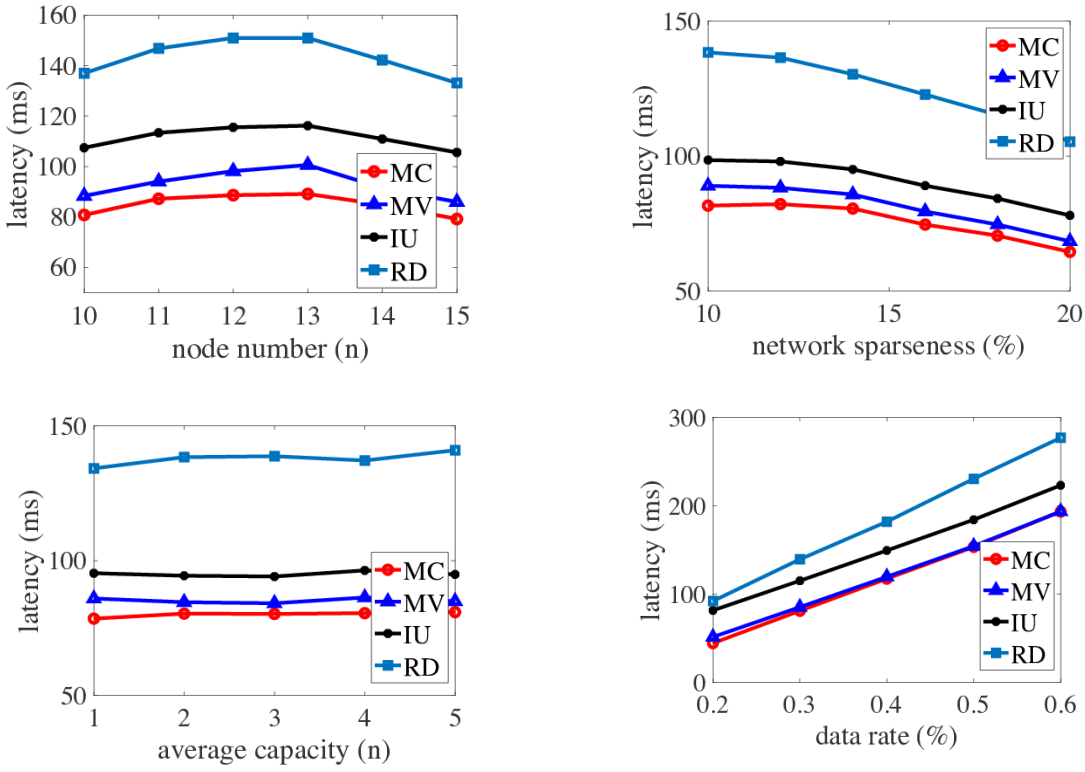


Figure 9: Performance comparison without data replication.

6.3 Algorithm Comparison

We compare four algorithms in the experiments:

• Random (RD) Algorithm. It is a benchmark algorithm. During the first m rounds, a type i data is randomly selected and placed. After that, a data is randomly selected and put into the network.

• Min-Volume (MV) Algorithm. The data whose placement leads to the minimal cost increase is selected. Specifically, in the first m rounds, if a data has been selected, it cannot be selected again.

• Iteration Updating (IU) Algorithm. The IU Algorithm places different data requests in an order so that the location that leads to the minimal cost increase is selected in each round.

• Min-Cost (MC) Algorithm. The MC Algorithm is proposed in this paper and it is explained in Section 4 when there is no data replication.

• Rounding (RO) Algorithm. The RO Algorithm is proposed in this paper and it is explained in Section 5.4.

In a special scenario without data replication, the IU algorithm doesn't work since each data always has one data copy, and is thus removed in this case. Besides, the RD algorithm is not necessary since the MC algorithm is optimal.

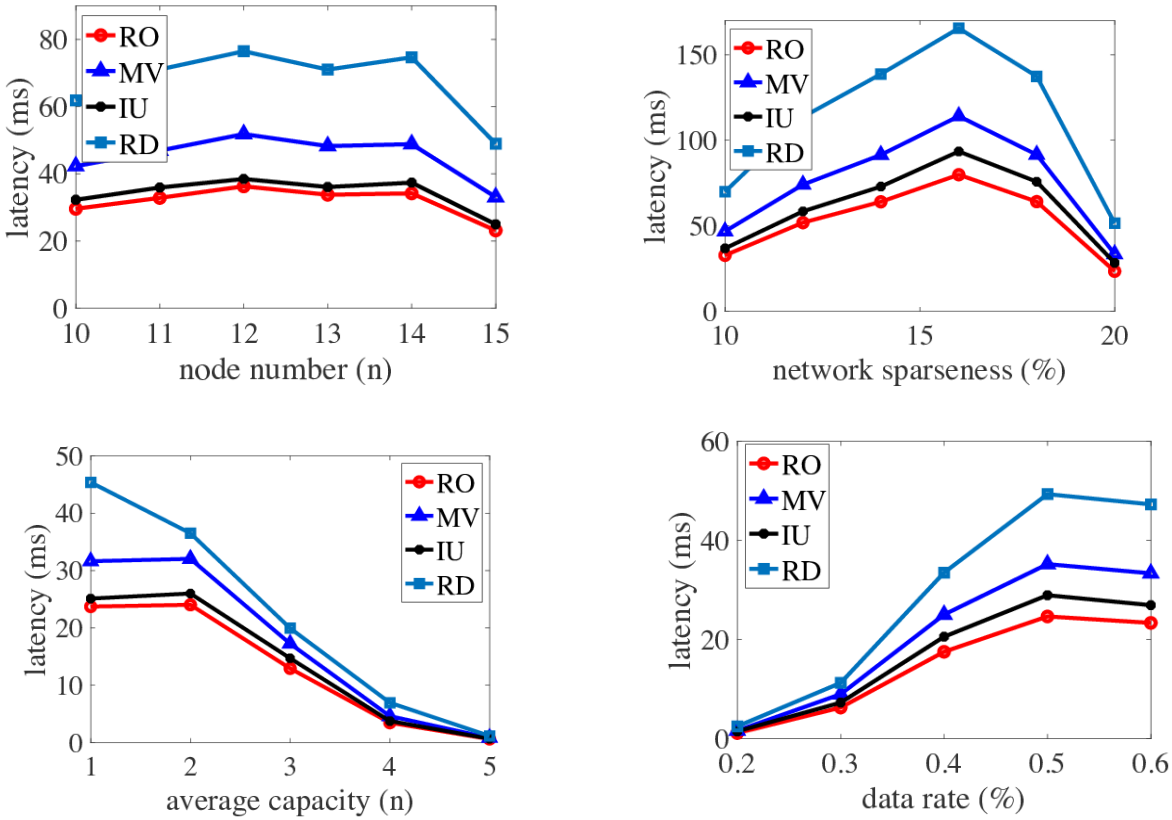


Figure 10: Performance comparison with data replication.

6.4 Experimental Results

6.4.1 Trace analysis

We verify the access latency between servers and their corresponding distances. The mapping result is shown in Fig. 10. We use three different distance measurement methods, that is, the shortest path which is the geo-distance of the corresponding GPS coordinates, the sum of latitude and longitude distances, and the area between two nodes in terms of latency estimation. Fig. 9 shows the cumulative distribution functions of three distance measurements, i.e., Fig. 9 shows that using the shortest-path has a relatively low estimation error, i.e., 10% for 80% of nodes when using the shortest path to estimate the latency between a pair of servers.

6.4.2 Results without data replication

Fig. 11 shows the results of the performance of proposed algorithms in the case, where there is no data replication. The results clearly show that the proposed MC algorithm achieves the lowest latency, followed by the MV algorithm. The RD algorithm's performance is the worst,

which demonstrates the necessity of data placement optimization. In Fig. 10, we change the number of servers in the experiments, i.e., the size of the FNs. The result shows that when the FNs have a larger size, improper data placement leads to bad performance. The RD algorithm has more than 100% of the MC algorithm's latency in Fig. 10. In Fig. 10, we gradually increase the average number of edges in the network with a certain amount of servers. The results show that when the network is sparse, there is a large performance difference between algorithms. In the experiments, the average latency decreases around 20% with a lower network spareness level. The IU algorithm has similar performance to MV since they are both greedy algorithms which cannot generate the optimal data placement order.

In Fig. 10, we change the average storage size of each server. The results show that the average latency is relatively stable. A possible reason is that the data is generated uniformly in the experiments and thus, the placement of data into different locations has minimum influence on the final result. Fig. 10 shows the results of different data request rates. When the data request rate increases, all algorithms have a larger latencies.

6.4.3 Results with data replication

Fig. 12 shows the performance of proposed algorithms in a case where there is data replication. The budget number is two times the number of data request in the experiments. The results clearly show that the proposed RO algorithm achieves the lowest latency, followed by the IU, MV, and RD algorithms. The RD algorithm's performance is the worst, which demonstrates the necessity of data placement optimization.

In Fig. 11, we change the number of servers in the experiments, i.e., the size of the FNs. The results show that when the FNs have a larger size, improper data placement leads to bad performance. The RD algorithm has more than 150% the latency of the IU algorithm in Fig. 11. Compared with the results in Fig. 10, all algorithms have better performance due to a greater amount of data placed in the network. In Fig. 11, we gradually increase the average number of edges in the network with a certain amount of servers. The results show that when the network is sparse, the performance difference between different algorithms is large. An interesting result is that the average latency first increases, then later decreases with an increase in the network sparseness level.

In Fig. 11, we change the average storage size of each server. The results show that along with an increase in average storage size, the latency decreases very quickly due to increased placement flexibility. In Fig. 10, the average latency is reduced by more than 20% with one additional storage capacity in the server node. Fig. 11 shows the results of different data request rates. When the data request rate increases, all algorithms have a larger latency. However, the RD algorithm has the fastest speed in terms of latency increasing, which demonstrates the effectiveness of the data placement optimization.

6.4.4 Summary

Based on the experiments, we find that data placement optimization is very necessary. The average performance can be improved by more than 50% in most cases by comparing the proposed algorithms with the random placement. In a general case with data replication, the proposed RO algorithm has significant improvement compared to other algorithms, which indicates that both the data budget distributions in each data request and their corresponding placements are very important.

7 Conclusion

In this paper, we consider the data placement issue in the Fog Networks (FNs) so that users have low access delay. Specifically, we discuss the Multiple Data placement with Budget Problem (MDBP), whose objective is to minimize the overall access latency. We begin with a simple case, where there is no data replication. In this case, we propose a min-cost flow transformation and thus find the optimal solution. We further propose efficient updating strategy to reduce the time complexity in the tree topology. In a general case with data replication, we prove that the proposed problem is NP-complete. Then, we propose a novel rounding algorithm, which incurs a constant-factor increase in the solution cost. We validate the proposed algorithm by using the PlanetLab trace and the results show that the proposed algorithms improve the performance significantly.

References

[1] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," ACM Transactions on Management Information Systems, vol. 6, no. 4, p. 13, 2016.

[2] "Facebook statistics, 2018." [Online]. Available: https://www.statista.com/statistics/264810/number-of- monthly-active-facebook-users-worldwide/

[3] "Cisco visual networking index: Forecast and methodology, 20162021." [Online]. Avail- able: https://www.cisco.com/c/en/us/solutions/collateral/service- provider/visual-networking-index-vni

[4] D. Ghose and H. J. Kim, "Scheduling video streams in video- on-demand systems: A survey," Multimedia Tools and Applica- tions, vol. 11, no. 2, pp. 167–195, 2000.

[5] M. Claeys, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latre ´, and F. De Turck, "An announcement-based caching approach for video-on-demand streaming," in Proceedings of the IEEE CNSM, 2015.

[6] G. Tang, K. Wu, and R. Brunner, "Rethinking cdn design with distributee time-varying traffic demands," in Proceedings of the IEEE INFOCOM, 2017.

[7] J. Sahoo, M. A. Salahuddin, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on replica server placement algorithms for content delivery networks," IEEE Communications Surveys & Tutorials, vol. 19, no. 2, pp. 1002–1026, 2016.

[8] N. Wang and J. Wu, "Minimizing the subscription aggregation cost in the content-based pub/sub system," in Proceedings of the IEEE ICCCN, 2016

[9] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," IEEE/ACM Transactions on Networking, vol. 8, no. 5, pp. 568–582, 2000.

[10] M. Yu, W. Jiang, H. Li, and I. Stoica, "Tradeoffs in cdn designs for throughput oriented traffic," in Proceedings of the ACM CoNEXT, 2012.

[11] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large- scale vod system," in IEEE/ACM Transactions on Networking, vol. 24, no. 4. ACM, 2010, pp. 2114–2127.

[12] S. Borst, V. Gupta, and A. Walid, "Distributed caching algo- rithms for content distribution networks," in Proceedings of the INFOCOM, 2010.

[13] L. Wang, R. Li, and J. Huang, "Facility location problem with different type of clients," Intelligent Information Management, vol. 3, no. 03, p. 71, 2011.

[14] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the internet," in Proceedings of the IEEE INFOCOM, 1999.

[15] J. Xu, B. Li, and D. L. Lee, "Placement problems for transparent data replication proxy services," IEEE Journal on Selected areas in Communications, vol. 20, no. 7, pp. 1383–1398, 2002.

[16] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Practical resource provisioning and caching with dynamic resilience for cloud- based content distribution networks," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 8, pp. 2169–2179, 2014.

[17] K. A. Kumar, A. Deshpande, and S. Khuller, "Data placement and replica selection for improving co-location in distributed environments," arXiv preprint arXiv:1302.4168, 2013.

[18] W. Ta ¨rneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmroth, "Dynamic application placement in the mobile cloud network," Future Generation Computer Systems, vol. 70, pp. 163–177, 2017.

[19] L. R. Ford Jr and D. R. Fulkerson, "A simple algorithm for finding maximal network flows and an application to the hitchcock problem," Tech. Rep., 1955.

[20] A. Goldberg and R. Tarjan, "Solving minimum-cost flow prob- lems by successive approximation," in Proceedings of the ACM STOC, 1987.

[21] L. Gao, H. Ling, X. Fan, J. Chen, Q. Yin, and L. Wang, "A popularity-driven video discovery scheme for the centralized p2p-vod system," in Proceedings of the IEEE WCSP, 2016.

[22] M. Charikar, S. Guha, E ´. Tardos, and D. B. Shmoys, "A constant-factor approximation algorithm for the k-median prob- lem," in Proceedings of the ACM SOTC, 1999.

[23] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," SIAM Journal on Com- puting, vol. 38, no. 4, pp. 1411–1429, 2008.

[24] C. Lumezanu and N. Spring, "Measurement manipulation and space selection in network coordinates," in Proceedings of the IEEE ICDCS, 2008.

[25] [Online]. Available: https://www.infobyip.com/ipbulklookup.php.

[26] [Online]. Available: https://www.maxmind.com.