Replica Placement in Content Delivery Networks with Stochastic Demands and M/M/1 Servers

Chenkai Yang[1], Liusheng Huang[2], Bing Leng[1], Hongli Xu[2], Xinglong Wang[1]
School of CS & Tech., Univ. of Science & Technology of China, Hefei, Anhui 230027, P.R. China
Suzhou Institute for Advanced Study, Univ. of Science & Technology of China, Suzhou, Jiangsu 215123, P.R. China
Email: [1]{ckyang, lengb, wxlong}@mail.ustc.edu.cn, [2]{lshuang, xuhongli}@ustc.edu.cn

***Abstract*—Content Delivery Network (CDN) is proposed for replicating data objects at multiple locations in the network and encounters vast potential for future development, as a result of which, a number of replica placement techniques have been proposed over the last decade. However, most of the existing works on replica placement (RP) ignore the statistical property of the demands and the restricted service rate of the servers. In this paper, we investigate the techniques of replica placement in CDNs with stochastic demands and M/M/1 servers to optimize the overall performance in the network. We first model the demands and the servers as independent Poisson streams and simple M/M/1 queueing systems, respectively. Then, a formal definition and formalization of RP problem will be given. We show that RP problem is NP-complete and propose two heuristic algorithms: Greedy Dropping (GD) and Tabu Search (TS). We conduct abundant simulation experiments to evaluate the performance of our proposed algorithms. According to our simulation results, both of the two algorithms are efficient in finding a feasible solution with high probability. Especially, the TS decreases the average delay of the demands about 50% on average.**

***Keywords*—Content Delivery Network (CDN), Replica Placement, Stochastic Demands, M/M/1 Queueing System**

I. Introduction

The growing number of Web applications has become an important motivation of improving Internet performance in recent years, as a solution to which, content delivery networks (CDNs) are systems used to improve accessibility and reliability by replicating data objects and caching them at multiple locations in the network [5]. The emergence of CDNs allows the content be replicated to the servers close to the clients, which results in fast and reliable Internet services [12]. With the success of commercial CDN applications, such as Akamai [1] and Limelight [2], CDNs have attracted more and more attention from both the industry and academic communities.

Since the Quality of Service (QoS) is tightly related to the locations of the replicas, Replica Placement (RP) algorithm has great impact on the overall performance in CDNs. According to the replication granularity, CDN systems can be categorized into two modes: full replication and partial replication [10]. A number of CDN replica placement techniques have been proposed on both full replication and partial replication mode over the last decade [14], [15], [17], [19], [22], [25], [27]. For example, Qiu *et al.* [19] have formulated the full replication as minimal K-median problem, in which the number of replicas was restricted. Several algorithms like greedy algorithm, tree based algorithm were proposed for solving the web server replica placement problem that optimizing an average access latency of all clients in the online CDNs. Except for full replication, Sun *et al.* [22] have proposed a new optimization model with server storage capacity constraints for the partial replication problems and designed an three-stage algorithm (CPM) to minimize the total cost in the network. Among all these works, Greedy Heuristic algorithms are considered to be the best algorithms.

However, all these works assume that the exact traffic demands from each client is known *a priori* and a demand will be responded once it arrives at a server. These assumptions are not realistic in many practical applications. The replica placement problem is typically solved during the server configuration stage, where the exact demand information is unavailable. Therefore, traffic demands must be modeled as stochastic quantities. A demand is typically an aggregation of many independent traffic sources, and by the central limit theorem, it can be approximated using a Poisson random variable [6]. Moreover, the service rate of each server is restricted. In general, a demand goes to the closest server who holds the corresponding object and waits for service in a queue. Thus, the waiting time for each demand should not be ignored in the system model. The servers could be modeled as simple M/M/1 queueing systems, which is common in related literatures (e.g., [7], [8]). Notes that the total delay for a demand should include the link delay and the waiting time in the queue at a server.

In this paper, we investigate the techniques of replica placement in CDNs with stochastic demands and M/M/1 servers to optimize the overall performance in partial replication mode. The motivation of our work is to build a more realistic model for the demands and servers in replica placement problem and achieve better overall performance in practical CDN systems. We model the demands for each object from a client as an independent Poisson stream. Furthermore, the servers are modeled as simple M/M/1 queueing systems, which means, the service time at each server obeys the exponential distribution. With the constraints of server capacity and waiting time upper bound, the objective is to find a replica placement scheme that minimizes the expected total number of demands traveling to and waiting at their closest server such that the overall performance of the CDN system could be optimized. We show such a problem is an NP-complete problem. Two heuristic algorithms include Greedy Dropping (GD) and Tabu Search (TS) are proposed. The main contributions of this paper are

978-1-4799-7575-4/14/$31.00 ©2014 IEEE

listed as follows:

1. We model the demands and the servers as Poisson streams and M/M/1 queueing systems respectively, based on which, we formulate the replica placement problem in CDNs and prove it is an NP-complete problem. To our best knowledge, this is the first work that deals with the problem of replica placement in CDNs with stochastic demands and M/M/1 servers.
2. As NP-hardness, we develop two heuristic algorithms include Greedy Dropping (GD) and Tabu Search (TS) to solve this problem. The Greedy Dropping is a greedy algorithm that removes objects from servers sequentially. The Tabu Search is a global searching algorithm that introduces a tabu list to avoid repetitive search.
3. The simulation results show that both of the two algorithms are efficient in finding a feasible solution with high probability. Especially, the Tabu Search reduce the average delay of the network about 50% compared with Random algorithm.

The remainder of the paper is organized as follows. Section II reviews the related works. In Section III, we introduce our system model and problem formulation. The detailed algorithm description of Greedy Dropping (GD) and Tabu Search (TS) are given in Section IV. Section V evaluates our proposed algorithms by simulations and Section VI concludes this paper.

II. Related Work

The concept of Content Delivery Network has been around for many years [11]. In this section, we'll review the related works about Replica Placement in CDNs. These works mainly fall into two categories: full replication and partial replication [9].

In full replication problem, the replica granularity is a mirror server. In fact, this problem is a variant of the facility location problem, which is also known as K-median problem [13]. As an example, in [16], the authors have proved that the replica placement problem in P2P networks has presented as a Clustered K-median problem, which was proven to be NP-complete. Besides, many other works on the replica placement in more complex conditions have been carried out. In [17], Li *et al.* settled the replica placement problem under a given traffic pattern. The drawback is that the proposed approach can only obtain the optimal solution for the tree topology. Qiu *et al.* also formulated the problem as minimal K-median problem in [19]. The difference is that the number of replicas was restricted in this work. However, the number of demands served by each replica was unbounded. The research conducted by Jamin *et al.* [20] is similar to [19]. Concluded from their work, despite of the placement scheme, increasing the number of replicas is only efficient when the number of replicas is very small. It suggested that fanout-based placement methods should work as well as greedy algorithms, which was used in [18] by exploring the effect of various replica placement approaches.

In large-scale Internet services (e.g. multimedia service), it is difficult for a single server to store the whole contents. As the Internet services are growing more and more abundant, partial replication is more in line with the actual in nowadays. The studies of Replica Placement are mainly focus on the partial replication in recent years [21], [22], [23], [24]. Our work is also belong to partial replication.

In [21], the authors formulated the object replication problem. Each replica is a copy of object. The authors shown that it is NP-complete and proposed several heuristic algorithms. Sun *et al.* [22] proposed a new optimization model with server storage capacity constraints for the replica placement problems. They divide the replica placement problems into two sub-problems: the number of each replica and which servers to store the replicas. An three-stage algorithm (CPM) was designed for allocating replicas to minimize the total cost in the network. Consider the unpredictable environments, static robust replica placement techniques was proposed for CDNs to support collaborative decision making in [23]. For this purpose, the authors proposed four different heuristics, each with different characteristics. Except for the centralized algorithms, many works have been done for distributed algorithms. In [24], the authors designed a distributed approximation algorithm by caching and replicating of most popular data objects, which results in reduction of network bandwidth usage and the access latency. Drwal *et al.* [32] formulate the optimal data placement in a variant, which combines connection, processing and storage costs. A decentralized algorithm was given, based on randomized rounding, which achieves an asymptotically logarithmic performance bounds with high probability.

All mentioned works above assume that the exact traffic demands from each client is known *a priori* and a demand will be responded once it arrives at a server. However, the exact demand information is unavailable in many applications. What's more, the service rate for the servers is restricted in reality. Thus, these existing works could not guarantee the performance of the CDNs in practical applications.

III. System Model and Problem Formulation

In this section, we'll introduce the system model and formulate the problem of replica placement, based on which, the NP-hardness will be proved.

A. System Model

Consider a generic CDN system consists of m clients and n servers. The clients and servers of the system are interconnected through a communication network. There are r different objects, each of which is denoted by O_k, for $\forall k \in R, R = \{1, 2, ..., r\}$. Multiple copies of an object can be replicated to different servers respectively. For simplicity, we assume that all the objects have the same size and the capacity of all servers are the same [9]. So, the storage capacity of each server can be represented by the maximum number of objects it can holds, which is denoted by $\overline{p}$. The link delay from client C_i ($\forall i \in M, M = \{1, 2, ..., m\}$) to server S_j ($\forall j \in N, N = \{1, 2, ..., n\}$) is denoted by d_{ij}. All these assumptions are often undertaken in formulating a replica placement problem [23].

The replica placement problem is typically solved during the server configuration stage, where the exact demand information is unavailable. Thus, traffic demands should be modeled as stochastic quantities. According to the statements in [6], demands could be approximated using a Poisson random

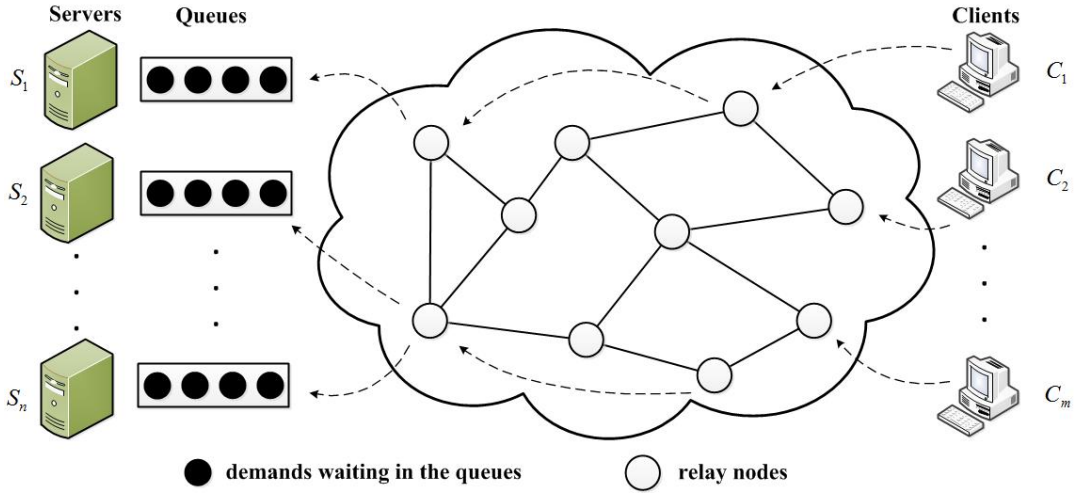


Fig. 1: The M/M/1 Queueing Model for Stochastic Demands from Clients

variable. We assume that demands for object O_k from client C_i form an independent Poisson stream with parameter λ_i^k, for $\forall i \in M, k \in R$. Moreover, the service rate of each server is restricted. The servers are modeled as simple M/M/1 queueing systems, that is, the service time at each server obeys the exponential distribution with common service rate μ. In our model, a demand goes to the closest server who holds the corresponding object and waits for service in a queue. To guarantee that each demand is served in time, there is an upper bound $\overline{W}$ for the expected waiting time of demands. The primary notations used in this paper are summarized in Table I.

TABLE I: PRIMARY NOTATIONS

Symbols	Descriptions
C	Set of the clients ($C = \{C_i : \forall i \in M\}$)
S	Set of the servers ($S = \{S_j : \forall j \in N\}$)
O	Set of the objects ($O = \{O_k : \forall k \in R\}$)
d_{ij}	Link delay from client C_i to server S_j
λ_i^k	Demand rate for object O_k from client C_i
γ_j	Demand rate at server S_j
μ	Common server rate for each server
W_j	Expected waiting time of demands assigned to server S_j
$\overline{W}$	Upper bound for the expected waiting time of demands
$\sigma = 1/\overline{W}$	Surplus service capacity to ensure $W_j \leq \overline{W}$
$\overline{p}$	Storage capacity of each server

The M/M/1 queueing model for stochastic demands from clients is illustrated in Fig.1. Each demand is transmitted to the closest server who holds the corresponding object via the relay nodes. Since the restricted processing capability of the servers, the demand may not be responded immediately. Therefore, the total delay for a demand consists of the link delay and the waiting time in the queue. We should note that the average delay of the demands is related to the total number of demands traveling in the network and waiting at their closest servers with the corresponding objects.

With the model of demands and servers, the replica placement problem can be stated as follows: given a client set with demand rates denoted by λ_i^k, a server set with common service rate μ and capacity $\overline{p}$, and an upper bound for the expected waiting time of demands $\overline{W}$, find a solution of replica placement that minimizes the expected total number of demands traveling to and waiting at their closest server with the corresponding object such that the total delay in the CDN system could be minimized.

B. Problem Formulation

Two decision variables are introduced to formulate the replica placement problem. The first one is a binary *n*-by-*r* matrix $\mathbf{y} = [y_{jk}]$, in which

$$y_{jk} = \begin{cases} 1, & \text{if a copy of } O_k \text{ is replicated on } S_j, \\ 0, & \text{otherwise,} \end{cases}$$

where $j \in N, k \in R$. The other is a binary three-dimensional matrix $\mathbf{x} = [x_{ijk}]$, which assumes

$$x_{ijk} = \begin{cases} 1, & \text{if demands for } O_k \text{ from } C_i \text{ is assigned to } S_j, \\ 0, & \text{otherwise,} \end{cases}$$

where $i \in M, j \in N, k \in R$.

The aggregate travel time of demands per unit time is

$$T = \sum_{i \in M} \sum_{j \in N} \sum_{k \in R} \lambda_i^k d_{ij} x_{ijk}.$$

Since each server behaves as an M/M/1 queueing system, the expected waiting time at server S_j is $W_j = 1/(\mu - \gamma_j)$ where $\gamma_j = \sum_{i \in M} \sum_{k \in R} \lambda_i^k x_{ijk}$ [4]. Thus, the aggregate waiting time of demands per unit time is

$$V = \sum_{i \in M} \sum_{j \in N} \sum_{k \in R} \lambda_i^k x_{ijk} W_j = \sum_{j \in N} \frac{\gamma_j}{\mu - \gamma_j}.$$

According to Little's Law [3], T represents the average number of traveling demands, while V is the average number of waiting demands in the queues.

To guarantee that each demand is assigned to the closest

server who holds the corresponding object, it requires that

$$\sum_{q\in N} d_{iq}x_{iqk} \le (d_{ij}-\Delta)y_{jk}+\Delta, \forall i\in M, j\in N, k\in R,$$

where Δ is a large positive number (e.g., $\Delta = \max\{d_{ij} : i \in M, j \in N\}$). When $y_{jk} = 0$, this constraint is invalid because Δ is large. When $y_{jk} = 1$, demands for object O_k cannot be assigned to a server which is farther than server S_j, otherwise, the constraint will be violated.

Based on the analysis above, we can obtain the following mathematical programming formulation:

$$\min \sum_{i\in M}\sum_{j\in N}\sum_{k\in R} \lambda_i^k x_{ijk}(d_{ij}+\frac{1}{\mu-\sum_{s\in M}\sum_{t\in R}\lambda_s^t x_{sjt}}),$$

subject to

$$\sum_{k\in R} y_{jk} \le \overline{p}, \qquad \forall j\in N, \tag{1}$$

$$\sum_{j\in N} x_{ijk} = 1, \qquad \forall i\in M, k\in R, \tag{2}$$

$$x_{ijk} \le y_{jk}, \qquad \forall i\in M, j\in N, k\in R, \tag{3}$$

$$\sum_{q\in N} d_{iq}x_{iqk} \le (d_{ij}-\Delta)y_{jk}+\Delta, \quad \forall i\in M, j\in N, k\in R, \tag{4}$$

$$\sum_{i\in M}\sum_{k\in R} \lambda_i^k x_{ijk} \le \mu-\sigma, \qquad \forall j\in N, \tag{5}$$

$$x_{ijk}\in\{0,1\}, y_{jk}\in\{0,1\}, \qquad \forall i\in M, j\in N, k\in R. \tag{6}$$

The objective is to minimize the sum of the expected number of traveling and waiting demands, subject to constraints (1) $\sim$ (6). Inequalities (1) assures for the number of objects replicated on each server not exceeding the capacity. Constraints (2) and (3) ensure that each demand is assigned, and only to an server who holds the corresponding object. To meet to the Constraint guarantee that the demand is assigned to the closest server with corresponding object, we add to the model constraint (4). Constraint (5) ensures that the expected waiting time at any server does not exceed $\overline{W}$. Finally, the last constraint (6) ensures the binary nature of the two decision variables $\mathbf{x}$ and $\mathbf{y}$.

As shown above, the main factors that lead to the hardness of this problem include the nonlinear objective function, the large number of constraints, and the binary nature of the decision variables. It is proved that the replica placement problem defined above is NP-complete in the following theorem.

Theorem 1: The replica placement problem (defined above) is an NP-complete problem.

Proof: We consider a special example of this problem. Assume that there is only one object O ($r = 1$) in the CDN system. The original problem is reduced to find locations for the object O. As a result, this is just a facility location problem with stochastic customer demand and immobile servers [4], which has been proved to be NP-hard. As the special modeled facility location problem is just a special case of our RP problem, RP is an NP-complete problem too. Thus, the NP-hardness is proved. ■

By the end of this section, some intuitive observations that may be helpful for our algorithms are stated as follows:

- There is a conflict between component T and V, which means increase of T may result in the decrease of V and vice versa;
- For a given value of μ, both T and V decrease when the value of $\overline{p}$ increases;
- For a given value of $\overline{p}$, both T and V decrease when the value of μ increases.

IV. Heuristic Algorithms

Due to NP-hardness, finding an optimal solution is infeasible. Instead we favor heuristics to tackle the problem. In this section, two heuristic algorithms are developed for the problem. The first one is the greedy dropping heuristic, which is a construction algorithm. The other is an improvement for the greedy dropping using tabu search. As the NP-hardness of the problem, the proposed heuristic approaches may fail to find a feasible solution for this problem.

A. Greedy Dropping Heuristic Algorithm

Generally speaking, this kind of problem can be solved by two kinds of construction heuristics. One is greedy adding algorithm that adds objects to servers sequentially. Another is greedy dropping that removes objects from servers sequentially. If we apply a greedy adding algorithm, we generally select next server to place an object that minimizes the objective function. However, it is infeasible for the problem we studied in this paper. If too few objects are placed, at least one of the queues at a server may be unstable, which will lead to an infinity value of the objective function. Hence, we focus on a greedy dropping heuristic algorithm instead.

Initially, we suppose each server holds all of the objects in regardless of the server capacity. Then we select one object on a server to remove, step by step. In general, after we remove an object, the objective function value will increase since the number of travel demands increases and the number of waiting demands may also increase. Hence, at each iteration of this heuristic, we remove an object from a server so that the objective function value will increase least. The procedure is repeated until one of the following stopping conditions is met:

1. No greater than $\overline{p}$ objects are left in each server, and further removing any object from a server provides a worse objective function value than the current one;
2. Greater than $\overline{p}$ objects are left in at least one of the servers, and any further removing causes the arrival rate of at least one server to exceed the upper bound $\mu-\sigma$.

In case 1 the procedure stops at a feasible solution, while in case 2 the procedure stops at an infeasible solution.

Since the decision variable $\mathbf{y}$ denotes the replica placement scheme, the corresponding objective function value could be denoted by $F(\mathbf{y})$. ($F(\mathbf{y}) = \infty$ when $\mathbf{y}$ is infeasible.) We define $\mathbf{y}\ominus(j,k)$ as the replica placement scheme that remove object O_k from server S_j on the basis of scheme $\mathbf{y}$, which means, set $y_{jk} = 0$ and keep other elements in $\mathbf{y}$ unchanged. The detailed algorithm description is shown in Algorithm 1.

Algorithm 1 Greedy Dropping (GD) Heuristic Algorithm

Step 1: Initialization
for each y_{jk} in $\mathbf{y}$ **do**
  $y_{jk} \leftarrow 1$ // each server holds all the objects
end for
Assign the demands according to the closest rule
Step 2: Iterations
while true **do**
  for each $y_{jk} = 1$ in $\mathbf{y}$ **do**
    $\rho_{jk} \leftarrow F(\mathbf{y} \ominus (j,k))$
  end for
  $(j_0, k_0) \leftarrow \arg\min\{\rho_{jk}, y_{jk} = 1\}$
  if $(\forall j \in N : \sum_{k \in R} y_{jk} \leq \overline{p})$ and $\rho_{j_0 k_0} \geq F(\mathbf{y})$ **then**
    break; // a feasible solution $\mathbf{y}$ has been found
  else if $(\exists j \in N : \sum_{k \in R} y_{jk} > \overline{p})$ and $\rho_{j_0 k_0} = \infty$ **then**
    break; // no feasible solution is found
  else
    $\mathbf{y} \leftarrow \mathbf{y} \ominus (j_0, k_0)$
  end if
end while

B. Tabu Search Heuristic Algorithm

In the greedy dropping heuristic, the algorithm stops at the first local optimum it reaches (when $\forall j \in N, \sum_{k \in R} y_{jk} \leq \overline{p}$) or at an infeasible solution (when $\exists j \in N : \sum_{k \in R} y_{jk} > \overline{p}$). In order to investigate a better solution other than the first one , or to possibly find a feasible solution in the case of GD heuristic fails, we propose a global searching algorithm that introduces a tabu list to avoid repetitive search.

It is necessary to find an initial solution for tabu search. Since the greedy dropping heuristic is a convenient way to generate the initial solution, it is applied firstly. If a feasible solution is obtained, it will serve as the initial solution for our tabu search. Otherwise, a greedy adding procedure that use only traveling component T as the objective function is applied. The greedy adding procedure starts with one copy of each object in the system and add objects sequentially. In each iteration, we add an object to a server that providing the largest decrease in expected traveling demands. This procedure terminates once there are just $\overline{p}$ objects in each server. The solution obtained here may be infeasible because of violating the waiting time upper bound. However, it will serve as the initial solution instead since it has a small traveling component and meets the constraint of the server capacity exactly.

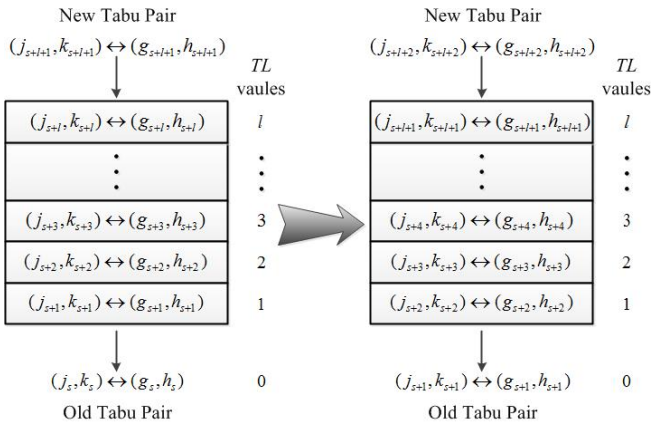


Fig. 2: Illustration of the Tabu List

After an initial solution is obtained, the tabu search procedure is implemented. We define the search neighborhood $S(\mathbf{y})$ of a solution $\mathbf{y}$ to be the set of all possible swaps including the swapping of different objects between two servers. In each iteration, we select the swap that provides the best objective function value. If all possible swaps in $S(\mathbf{y})$ lead to an infeasible solution, we select the one with the least number of servers that violating the waiting time upper bound. The search procedure will terminate until H successive swaps have no improvement on the objective function value, where H is a pre-defined positive integer. What's more, a tabu list is generated that marks a subset of possible swaps as forbidden in order to prevent the search from repeating executed swaps in the recent past. The tabu list is updated during each iteration.

Algorithm 2 Tabu Search (TS) Heuristic Algorithm

Step 1: Initialization
Let $\mathbf{y}$ be the solution generated by the greedy dropping
if $\mathbf{y}$ is infeasible **then**
  $\mathbf{y} \leftarrow Greedy\_Adding(T)$ // use greedy adding involving only traveling component to find an initial solution
end if
$\mathbf{y}^* \leftarrow \mathbf{y}$, $h \leftarrow 0$
for each j, g in N and k, h in R **do**
  $TL((j,k) \leftrightarrow (g,h)) \leftarrow 0$
end for
Step 2: Iterations
while $S(\mathbf{y}) \neq \emptyset$ **do**
  if all the swaps in $S(\mathbf{y})$ are infeasible **then**
    Let $(j_0, k_0) \leftrightarrow (g_0, h_0)$ be the swap in $S(\mathbf{y})$ with the least number of servers that have a waiting-time violation
  else
    Let $(j_0, k_0) \leftrightarrow (g_0, h_0)$ be the swap in $S(\mathbf{y})$ that provides the best objective function value
  end if
  if $F(\mathbf{y}[(j_0, k_0) \leftrightarrow (g_0, h_0)]) < F(\mathbf{y})$ **then**
    $h \leftarrow 0$, $\mathbf{y}^* \leftarrow \mathbf{y}[(j_0, k_0) \leftrightarrow (g_0, h_0)]$
  else
    $h \leftarrow h + 1$
  end if
  if $h = H$ **then**
    break; // perform H successive swaps without improvement of the objective function value
  else
    $\mathbf{y} \leftarrow \mathbf{y}[(j_0, k_0) \leftrightarrow (g_0, h_0)]$
    $TL((j_0, k_0) \leftrightarrow (g_0, h_0)) \leftarrow L$
    $TL((j,k) \leftrightarrow (g,h)) \leftarrow \max\{TL((j,k) \leftrightarrow (g,h)) - 1, 0\}$, $\forall (j,k) \leftrightarrow (g,h) \neq (j_0, k_0) \leftrightarrow (g_0, h_0)$
  end if
end while

To present the algorithm, we introduce the following additional notations:

- $\mathbf{y}^*$: the best solution found so far;
- L: length of the tabu list;

Algorithm 3 Greedy_Adding (T)

Input: T
// T is the objective function;
Output: $\mathbf{y}$

for each y_{jk} in $\mathbf{y}$ **do**
  $y_{jk} \leftarrow 0$ // initialize each server to be empty
end for
Assign one copy of each object to the servers to minimize the objective function T
while true **do**
  for each $y_{jk} = 0$ in $\mathbf{y}$ **do**
    $\rho_{jk} \leftarrow T(\mathbf{y} \oplus (j,k))$
  end for
  $(j_0, k_0) \leftarrow \arg\min\{\rho_{jk}, y_{jk} = 0, \sum_{h \in R} y_{jh} < \overline{p}\}$
  if $\forall j \in N : \sum_{h \in R} y_{jh} = \overline{p}$ **then**
    break; // the capacity is reached
  else
    $\mathbf{y} \leftarrow \mathbf{y} \oplus (j_0, k_0)$
  end if
end while

- K: the maximum number of successive non-improvement swaps permitted in iterations;
- $(j,k) \leftrightarrow (g,h)$: a pair of swap between object O_k on server S_j and object O_h on server S_g;
- $\mathbf{y}[(j,k) \leftrightarrow (g,h)]$: a solution that swap object O_k on server S_j with object O_h on server S_g based on $\mathbf{y}$;
- $TL((j,k) \leftrightarrow (g,h))$: an integer used to record the tabu status of the swap between object O_k on server S_j and object O_h on server S_g;
- $\mathbf{y} \oplus (j,k)$: a solution that remove object O_k from server S_j on the basis of $\mathbf{y}$.

The frame of the tabu list is illustrated in Fig.2. The TL value of each forbidden item is set to L once it is added to the tabu list and will minus one in each iteration. Note that if $TL((j,k) \leftrightarrow (g,h)) > 0$, pair $((j,k) \leftrightarrow (g,h))$ is in the tabu list and cannot be used for swapping in the next $TL((j,k) \leftrightarrow (g,h))$ iterations. Based on the introduced notations, the set of feasible swaps is defined as $S(\mathbf{y}) = \{((j,k) \leftrightarrow (g,h)) : TL((j,k) \leftrightarrow (g,h)) = 0, y_{jk} = 1, y_{jh} = 0, y_{gh} = 1, y_{gk} = 0\}$. The detailed description of Tabu Search Heuristic is shown in Algorithm 2, while the greedy adding sub-procedure used in TS heuristic is described in Algorithm 3.

Since the GD heuristic is a sub-procedure of the TS heuristic algorithm, the TS heuristic algorithm will always spend more time than the GD heuristic. However, the solution provided by TS will be no worse than GD heuristic anyway.

V. Numerical Results

This section mainly presents the numerical results to demonstrate the efficiency of GD and TS heuristic algorithms. What's more, we conduct comparisons between our proposed algorithms and random approach. We first introduce the simulation settings.

A. Simulation Settings

The network topology we used in the simulations are generated by the GT-ITM topology generator toolkit [28], which has been successfully used in recently published researches to simulate large-scale Internet topologies, e.g. [29], [30]. The simulated network has a total of 1000 nodes. Out of the 1000 nodes, we select at random: (a) 150 to be the clients and (b) 30 to be the servers that hold the replicas of 200 data objects. The rest of the 820 nodes are designated to be relay nodes. The link delay on each link is set to be 5 ms. Based on the network topology and the link delay, we can calculate the link delay d_{ij} between each pair of server and client. The demand rate matrix $[\lambda_i^k]$ is generated randomly under the constraint that the total demand rate from each client do not exceed 40.

In our computational simulations, two important parameters are under study:

- the capacity of servers $\overline{p}$, with seven levels 30, 35, 40, 45, 50, 55, 60;
- the common service rate of servers μ, with three levels 300, 330, 360.

Therefore, we have a total of 7×3 scenarios. To have enough confidence in our simulation results, the locations of the clients and servers are altered ten times per scenario. This results a total of 210 runs for each algorithm. The length of the tabu list in the TS heuristic is set to $L = 10$, and the maximal number of non-improvement swaps allowed is set to $H = 15$. The waiting time upper bound is set to $\overline{W} = 0.4$. To evaluate the performance of our approaches in the stochastic networks, we compare the two algorithms with the randomized approach, in which replicas of the objects are assigned to servers randomly restrained by the server capacity $\overline{p}$.

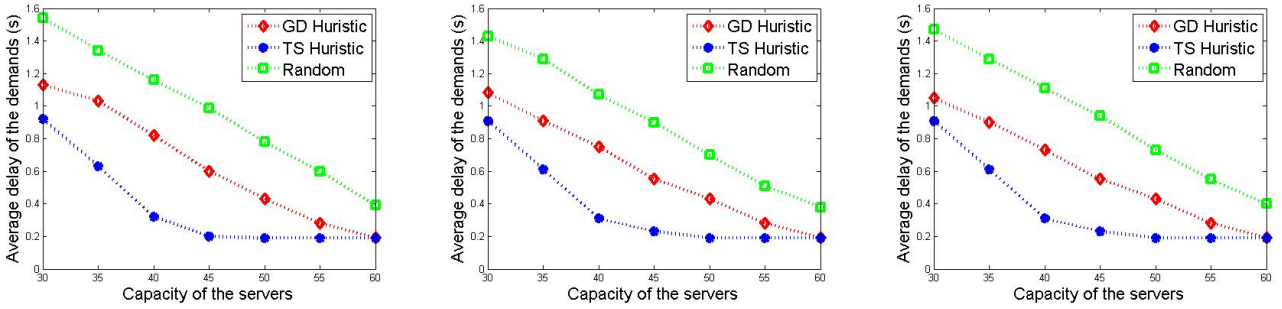
Heuristic algorithms were programmed in Microsoft Visual Studio 2010. All the programs were run on a Lenovo M4360 PC with 4G RAM and 3.2GHz CPU. Computation times are in seconds.

B. Simulation Results

We'll identify the performance of the proposed two heuristic algorithms. Besides, we also employ Randomized approach in each scenario to compare with our algorithms. In our experiments, we use the average delay for the demands as the first criterion to evaluate the algorithms. Notes that the objective function value in our formulation is proportional to the average delay, which directly reflects the effect of the solutions found by our algorithms. Since the algorithms may fail to find a feasible solution, the failure solutions are not calculated in the average delay.

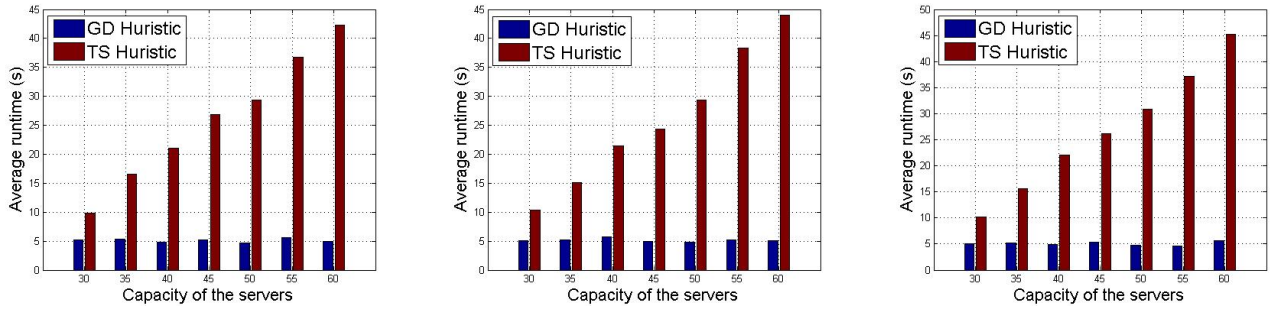
The results are shown in Figure 3. In each sub-figure, the red curve represents the average delays of GD heuristic, while the blue curve represents the average delays of TS heuristic. As a comparison, the performance of randomized approach is illustrated by the green curves. From the results shown in Figure 3, we can see that

- Generally: (i) the TS heuristic outperforms other approaches in all scenarios; (ii) the performance of GD heuristic is better than that of randomized approach



(a) Average Delay vs. Capacity of Servers ($\mu = 300$) (b) Average Delay vs. Capacity of Servers ($\mu = 330$) (c) Average Delay vs. Capacity of Servers ($\mu = 360$)

Fig. 3: Average Delay vs. Capacity of Servers in different values of μ



(a) Average Runtime vs. Capacity of Servers ($\mu = 300$) (b) Average Runtime vs. Capacity of Servers ($\mu = 330$) (c) Average Runtime vs. Capacity of Servers ($\mu = 360$)

Fig. 4: Average Runtime vs. Capacity of Servers in different values of μ

in each scenario; (iii) our approaches can improve the performance significantly in scenarios.

- In each sub-figure, the average delay trends down along with the capacity of the servers $\overline{p}$. The reason for this is that when $\overline{p}$ is larger, the density of the servers that hold one object increases, which results in the decrease of link delay.
- The average delay of GD is slightly influenced by the service rate μ when $\overline{p}$ is small. Nonetheless, the performance of TS is not significantly affected by μ regardless of $\overline{p}$.
- The average delay of the GD heuristic is about 65% of the randomized approach on average, while the TS heuristic decreases the average delay about 50% on average.

The proposed algorithms may fail to find a feasible solution, so the success rate is an important criterion to evaluate the algorithms. Since the result is related to the service rate μ, we count the successful runs in different values of μ. The statistic result of the success rate is listed in Table II. As shown in the table, the success rate of TS is higher than GD on average. The reason for this may be that GD is a sub-procedure of the TS heuristic. Nevertheless, both the GD and TS heuristics can find feasible solutions in all runs when the service rate μ is large enough. In general, both the GD and TS approaches can find a feasible solution with probabilities higher than 90%.

Except for average delay and success rate, the average runtime is also an important criterion to evaluate our algorithms. Figure 4 shows the relationship between average runtime and capacity of servers. In each sub-figure, the blue bars represent the average runtime of GD heuristic in different scenarios, while the red bars represent the average runtime of TS heuristic. Just like in the evaluation of average delay, runs that fail to find a feasible solution are not counted in our calculation. We can conclude from the Figure 4 that

TABLE II: Success Rate of the Heuristics

Service Rate μ	GD Heuristic	TS Heuristic
300	92.86%	97.14%
330	98.57%	100%
360	100%	100%

- The runtime of the TS heuristic is always larger than that of the GD heuristic. On average, the TS heuristic spends over four times as long as the GD heuristic. This is resulted by the fact that GD is a sub-procedure of the TS heuristic.
- The runtime of the GD heuristic is not significantly affected by the different values of $\overline{p}$ and μ. The reason for this might be that $\overline{p}$ is too small in proportion to the number of objects.
- The runtime of the TS heuristic increases in $\overline{p}$, but is not significantly affected by the different values of μ. The reason for this is that the number of possible swaps in each iteration is larger when $\overline{p}$ is larger.

From experimental conclusions above, both the GD heuris-

tic and the TS heuristic are efficient in finding a feasible solution with high probability. What's more, solutions found by both of the two algorithms result much less average delay than the randomized approach. Especially, the TS decreases the average delay of the demands about 50% on average. However, it may take much more time for the TS heuristic to find a feasible solution with higher possibility.

VI. Conclusions

In this paper, we investigate the techniques of replica placement in CDNs with stochastic demands and M/M/1 servers to optimize the overall performance. We model the demands for each object from a client as an independent Poisson stream. Furthermore, the servers are modeled as simple M/M/1 queueing systems, that is, the service time at each server obeys the exponential distribution. With the constraint of server capacity and waiting time upper bound, the objective is to find a replica placement scheme that minimizes the expected total number of demands traveling to and waiting at their closest server such that the overall performance of the CDN system could be optimized. We show such a problem is a NP-complete problem. Two heuristic algorithms include Greedy Dropping (GD) and Tabu Search (TS) are proposed. Simulation results show that both of the two algorithms are efficient in finding a feasible solution with high probability. Especially, the TS decreases the average delay of the demands about 50% on average.

Acknowledgment

This paper is supported by the National Science and Technology Major Project under Grant No. 2012ZX03005-009, National Science Foundation of China under Grant No.U1301256, 61472383, 61170058, 61272133, and 61228207, Special Project on IoT of China NDRC (2012-2766), Research Fund for the Doctoral Program of Higher Education of China No. 20123402110019, the NSF of Jiangsu Province No. BK2012632, and Suzhou Industry Fund No. SYG201302.

References

[1] Akamai, "http://www.akamai.com."

[2] Limelight, "http://www.limelight.com/."

[3] D. Gross and C.M. Harris, *Fundamentals of Queueing Theory* [M], Wiley: New York, 1998.

[4] Q. Wang, R. Batta and C.M. Rump, "Algorithms for a Facility Location Problem with Stochastic Customer Demand and Immobile Servers," *Annals of Operations Research*, Vol.111, NO.1-4, Pages 17-34, 2002.

[5] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Computer Magazine*, Vol.7, NO.6, Pages 68-74, Nov./Dec. 2003.

[6] M. Johnston, H.W. Lee and E. Modiano, "Robust Network Design for Stochastic Traffic Demands," *Journal of Lightwave Technology*, Vol.31, NO.18, Pages 3104-3116, SEP.15 2013.

[7] B.K. Kumar and D. Arivudainambi, "Transient solution of an M/M/1 queue with catastrophes," *Computers & Mathematics with Applications*, Vol.40, NO.10-11, Pages 1233-1240, Nov./Dec. 2000.

[8] H. Jamjoom, C.T. Chou and K.G. Shin, "The impact of concurrency gains on the analysis and control of multi-threaded Internet services," in *Proc. of IEEE INFOCOM*, 2014.

[9] Z. Xu and L. Bhuyan, "QoS-Aware Object Replica Placement in CDN networks," in *Proc. of IEEE GLOBECOM*, 2005.

[10] C. Xu, *Scalable and Secure Internet Services and Architecture* [M], Detroit: Chapman and Hall/CRC, 2005: 28-30.

[11] F. Douglis and M.F. Kaashoek, "Scalable Internet Services," *IEEE Internet Computing*, Vol.5, NO.4, Pages 36-37, 2001.

[12] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, Vol.13, NO.7, Pages 50-58, Sep./Oct. 2002.

[13] D.S. Hochbaum, *Approximation Algorithms for NP-Hard Problems* [M], Course Technology: Berkeley, 1996.

[14] X. Tang J. Xu, "On Replica Placement for QoS-Aware Content Distribution," in *Proc. of IEEE INFOCOM*, 2004.

[15] S.U. Khan and I. Ahmad, "Comparison and analysis of ten static heuristicsCbased Internet data replication techniques," *Journal of Parallel and Distributed Computing*, Vol.68, NO.2, Pages 113-136, Feb. 2008.

[16] J. Zhou, X. Zhang, L. Bhuyan and B. Liu, "Clustered KCenter: Effective Replica Placement in Peer-to-Peer Systems," in *Proc. of IEEE conference on Global Telecommunications*, 2007.

[17] B. Li, M. Golin, G. Italiano, X. Deng and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet," in *Proc. of IEEE INFOCOM*, 1999.

[18] P. Radoslavov, R. Govindan and D. Estrin, "Topology-informed Internet Replica Placement," in *Proc. of the 6th International Workshop on Web Caching and Content Distribution*, 2000.

[19] L. Qiu, V.N. Padmanabhan and G.M. Voelker, "On the Placement of Web Server Replicas," in *Proc. of IEEE INFOCOM*, 2001.

[20] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt and L. Zhang, "On the Placement of Internet Instrumentation," in *Proc. of IEEE INFOCOM*, 2000.

[21] J. Kangasharju, J.W. Roberts and K.W. Ross, "Object replication strategies in content distribution networks," in *Proc. of the 6th Web Caching and Content Distribution Workshop*, 2001.

[22] J. Sun, S.Gao, W.Yang and Z.Jiang, "Heuristic Replica Placement Algorithms in Content Distribution Networks," *Journal of Networks*, Vol.6, NO.3, Pages 416-423, Mar. 2011.

[23] S.U. Khan, A.A. Maciejewski and H.J. Siegel, "Robust CDN replica placement techniques," in *IEEE International Symposium on Parallel and Distributed Processing*, 2009.

[24] S. Zaman and D. Grosu, "A Distributed Algorithm for the Replica Placement Problem," *IEEE Trans. on Parallel and Distributed Systems*, Vol.22, NO.9, Pages 1455-1468, Sep. 2011.

[25] M. Karlsson and M. Mahalingam, "Do we need replica placement algorithms in content delivery networks," in *Proc. of the 7th International Workshop on Web Caching and Content Distribution*, 2002.

[26] K. Kalpakis, K. Dasgupta and O. Wolfson, "Optimal placement of replicas in trees with read, write, and storage costs," *IEEE Trans. on Parallel and Distributed Systems*, Vol.12, NO.6, Pages 628-637, Jun. 2001.

[27] F.L. Presti, N. Bartolini and C. Petrioli, "Dynamic Replica Placement and User Request Redirection in Content Delivery Networks," in *Proc. of IEEE ICC*, 2005.

[28] Gtitm, "http://http://www.cc.gatech.edu/projects/gtitm/."

[29] Z. Li and P. Mohapatra, "The impact of topology on overlay routing service," in *Proc. of IEEE INFOCOM*, 2014.

[30] R. Gill, R. Paul and L. Trajkovic, "Effect of MRAI Timers and Routing Policies on BGP Convergence Times," in *Proc. of IEEE IPCCC*, 2012.

[31] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Communications of the ACM*, Vol.49, NO.1, Pages 101-106, Jan. 2006.

[32] M. Drwal and J. Jozefczyk, "Decentralized Approximation Algorithm for Data Placement Problem in Content Delivery Networks," in *3rd IFIP/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems*, 2012.