Optimization Algorithms for Proxy Server Placement In Content Distribution Networks

Jun Wu, Kaliappa Ravindran

Department of Computer Science, City University of New York, U.S.A.
Email: jwu@ccny.cuny.edu, cskar@cs.ccny.cuny.edu

Abstract--Popular Web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. One of the solutions is to using content distribution network (CDN). When we design a CDN we need to find proxy server placement to provide its clients with the best available performance while consuming as little resource as possible. This is an optimization problem. Among the solutions greedy algorithm yields better result with low computational cost. The drawback is that it is easy to trap in the local optimum. We propose genetic algorithm to solve this problem. We mathematically model the optimization problem and then give details about how to apply genetic algorithm to proxy server placement problem. Simulation results for a simple topology are presented for both greedy algorithm and genetic algorithm.

I. Introduction

With the explosive growth of the World Wide Web, popular Web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. Content distribution network (CDN) has been used for this purpose; it is a group of geographically dispersed servers deployed to facilitate the distribution of information generated by web publishers in a timely and efficient manner. Commercialized services have been launched such as Akamai [1], Digital Island [2] and Exodus [3].

CDN has two advantages. The first advantage is that it can reduce Client's latency and network traffic by redirecting client requests to a proxy server closest to that client. The second advantage is that it can also improve the availability of the system, as the failure of one proxy server does not result in entire service outage. Both advantages result from the fact that making multiple copies of the content of a Web publisher and distributing that content across the Internet removes the necessity for customer requests traversing a large number of routers to directly access the facilities of the Web publisher. This reduces traffic routed through the Internet as well as the delays associated with the routing of traffic.

When we design a CDN we need to find proxy server placement to provide its clients with the best available performance while consuming as little resource as possible. In a sense, we are dealing with an optimization problem [4]. Some research works have been done to solve this problem. Include [5], [6], [7], [8], [9], [10]. Among them, greedy algorithm [6] can yield result close to optimal at low computational cost. Its drawback is that it cannot guarantee to get the global optimum. It's easy to trap in the local optimum.

In this paper we propose using genetic algorithm to solve this problem. The rest of the paper is as follows. The proxy server placement problem in CDN is explained and the mathematical model is given in section II. In section III we will discuss in detail about greedy algorithm. In section IV we will give the detail about how to using genetic algorithm to place proxy servers. Simulation results for a simple topology are presented for both greedy algorithm and genetic algorithm in section V. Section VI is the future studying.

II. Mathematical model for optimizing proxy server placement

The infrastructure-level topology of a CDN is modeled as a graph G(V,E), where V is the set of in-network processing nodes and E is the set of edges interconnecting these nodes by network links. The clients reside in a set of nodes Vc, and access the content hosted on a remote master server node M. All the remaining nodes in the topology (V-M-Vc) are deemed as proxy-capable. A proxy placement algorithm chooses one or more of these nodes to host the proxies of M in a way to service the clients in a cost-optimal manner. The nodes hosting the proxies, Vr, are part of a content distribution tree

978-1-4244-3924-9/09/$25.00 © 2009 IEEE

T({M,Vc,V'},E') with the root node at M and the leaf nodes at Vc. The tree T is basically an overlay set up on the infrastructure topology G (V,E) with a set of edges E', where $E' \subseteq E$, $V' \subseteq (V-\{M,Vc\})$ and $Vr \subseteq V'$.

Given a server-client configuration {M,Vc} placed in the infrastructure topology G(V,E), we assume that the CDN service provider (SP) first runs an overlay multicast routing protocol, such as the variants of DVMRP (RFC 1075) and MOSPF (RFC 1584), to set up the tree T({M,Vc,V'},E'). At the routing level, the tree T may be globally optimal or incrementally optimal based on the tree setup algorithm chosen, where the optimality is measured in terms of the total number of hops in the interconnection paths between Vc and M.

After the overlay tree T is set up (regardless of how optimal T is at the routing level), the SP runs a proxy placement algorithm to select the nodes $Vr \subseteq V'$ to host the proxies in a cost-optimal manner. With the proxies Vr placed on the tree T, each client has exactly one path to reach its content serving node: which may be a proxy for M or the master server M itself. Here, the proxy placement determines the path length faced by various clients in accessing content, and hence impacts the QoS experienced in the CDN service interface to the clientele, namely, the content access latency experienced by individual clients.

The optimality of proxy placement is prescribed in terms of a cost metric associated with content distribution to the clients by the SP. One possible metric is the total bandwidth consumed for content delivery across all clients. Another metric may be the QoS degradation experienced by individual clients due to excessive content access latency (which translates to a revenue loss suffered by the SP). All such cost metrics are computable from the length of path segments from various clients to their content serving nodes and/or the path characteristics (such as link delays & capacity, link congestion, etc). For simplicity we define our object function as the total network transfer cost. We only consider content movement, because the control message overhead can be neglected compare to the overall cost of content movement.

We have I server in the network. $Server_i$, i = 1, 2, …, I and has K clients that request objects at (aggregate) rate r_k where k = 1, 2, …, K. We have the following variables: $x_i = 1$ if replica has been placed at server i, $x_i = 0$ otherwise. The constrain is that only J servers are allowed to place replicas.

The goal is to choose the x_i's so that a given performance metric is minimized. For simplicity, our goal is to minimize the average number of hops that a request must traverse. This reflects the download time of an object to some degree and can thus be used as an indicator of the user perceived latency.

We denote the vector of all x_i's by $\mathbf{X}$. Furthermore, we assume that each object j is initially placed on an origin server; we denote by O_j. We assume that all of the objects are always available in their original servers, regardless of the placement $\mathbf{X}$. We denote the placement of objects to origin servers as $\mathbf{X}_o$.

The number of hops that a request must traverse from client k is $d_k(\mathbf{X})$ where $d_k(\mathbf{X})$ is the shortest distance to a copy of a replica from client k under the placement $\mathbf{X}$. This nearest copy is either in the origin server O or in another proxy server where the object has been replicated. We assume that the client is always redirected to the nearest copy.

Let $R = \sum_{k=1}^{K} r_k$ be the total request rate of all clients. The average number of hops from all clients is then

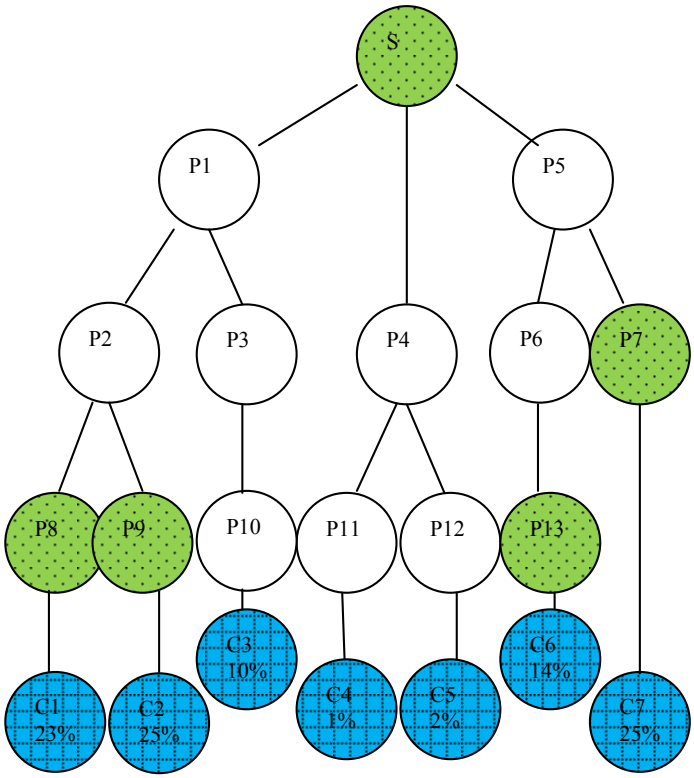
$$C(\mathbf{X}) = \frac{1}{R} \sum_{k=1}^{K} C_k(\mathbf{X}) = \frac{1}{R} \sum_{k=1}^{K} r_k d_k(\mathbf{X})$$

$$= \sum_{k=1}^{K} s_k \, d_k(\mathbf{X})$$

Where $s_k = r_k / R$. The placement $\mathbf{X}$ is constrained by the total number of proxy server is no more than J.

This cost function represents the long term average cost. For a large number of clients and potential proxy sites, it is not feasible to solve this problem optimally.

In Fig. 1 we have one master server, 7 clients, 13 potential proxy sites. Using exhaustive search to place m replicas in n potential sites the computational cost is n!/(m!(n-m)!.



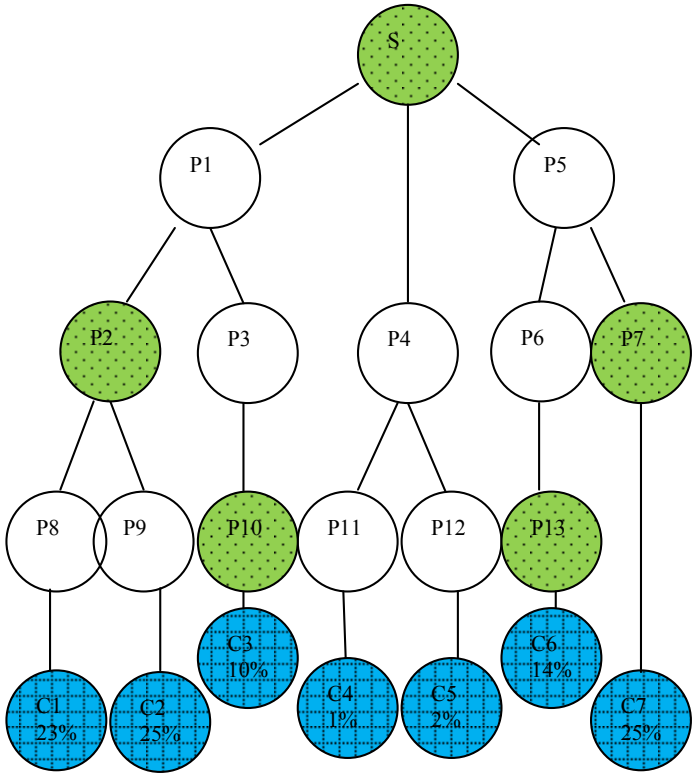
S: master server
P: potential proxy site
C: client
%: request rate

place 4 replicas out of 13 potential sites: 13*12*11*10/(4*3*2*1)
cost: 0.36
place m replicas out of n potential sites: n!/(m!*(n-m)!)

Fig.1. Using exhaustive search to place replicas

III. Proxy Server Placement by Greedy Algorithm

Qiu et al. [2001] propose a greedy algorithm. In each iteration, the algorithm selects one server, which offers the least cost, where cost is defined as the average distance between the server and its clients. In the *i*th iteration, the algorithm evaluates the cost of hosting a replica at the remaining $N - i + 1$ potential sites in the presence of already selected $i - 1$ servers. The computational cost of the algorithm is $O(N^2K)$. The authors also present a hot-spot algorithm, in which the replicas are placed close to the clients generating most requests. The computational complexity of the hot-spot algorithm is $N^2 + min(NlogN, NK)$. The authors evaluate the performance of these two algorithms and compare each one with the algorithm proposed in Li et al. [5]. Their analysis shows that the greedy algorithm performs better than the other two algorithms and its performance is only 1.1 to 1.5 times worse than the optimal solution. The authors note that the placement algorithms need to incorporate the client topology information and access pattern information, such as client end-to-end distances and request rates. Qiu's algorithm is for the replica placement. Similarly we can apply this algorithm in proxy server placement. The procedure is the same, but each client connects to a proxy being placed by the algorithm or to the master server, whichever is the closest. The drawback of this algorithm is that it can not guarantee to get the global optimum. It's easy to trap in the local optimum. Fig.2 show the result of the greedy algorithm.The cost 0.54 is suboptimal compare to the global optimum cost 0.36 (Fig. 1).



Choose 4 proxies: P2, P7, P10, P13
Cost: 0.54

Fig.2. Greedy algorithm finds the suboptimal

IV. Using Genetic Algorithm to Solve the Proxy Placement Problem

A genetic algorithm [11] (or short GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

We will demonstrate GA's design in detail, by presenting our proxy server placement encoding mechanism and then the selection, crossover, mutation and termination operators.

A chromosome encoding a replication scheme is a string consisting of M digits. Each digit is the label for the site chosen for proxy placement.

Fitness value f for proxy server placement problem: $\sum_{i=1}^{K} s_k \, d_k \, (X)$

where $s_k = r_k / R$. $R = \sum_{i=1}^{K} r_k$ be the total request rate of all clients. d_k (**X**) is the shortest distance to a proxy server from client k under the placement **X**.

Pseudo-code

- Choose initial population
- Evaluate the fitnesses of individuals in the population

Repeat

- Select best-ranking individuals to reproduce
- Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
- Evaluate the individual fitnesses of the offspring
- Replace worst ranked part of population with offspring

Until terminating condition

Initialization

The initial population is generated in a total random way.

Reproduction

We use one-point crossover. A crossover point on the parent organism string is selected. All data beyond that point in the organism string is swapped between the two parent organisms. Mutation is performed by simply flipping every digit with a certain probability u, known as the mutation rate. Although mutation is not the primary search operation and some algorithms omit it, it is very useful in our design. The reason is that in order for our algorithm to have practical applications it should achieve good performance when tackling very large chromosomes. A one point-crossover alone would not be able to explore the solution space fast while a multiple point crossover would result in highly disruptive.

During each successive epoch, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Selection methods rate the fitness of each solution and preferentially select the best solutions.

Fitness function is stochastic designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions.

Termination

This generational process is repeated until a termination condition has been reached. The termination condition here is the highest ranking solution's fitness has reached a plateau such that successive iterations no longer produce better results.

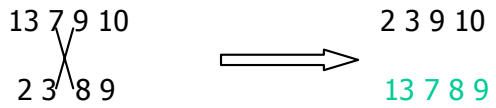
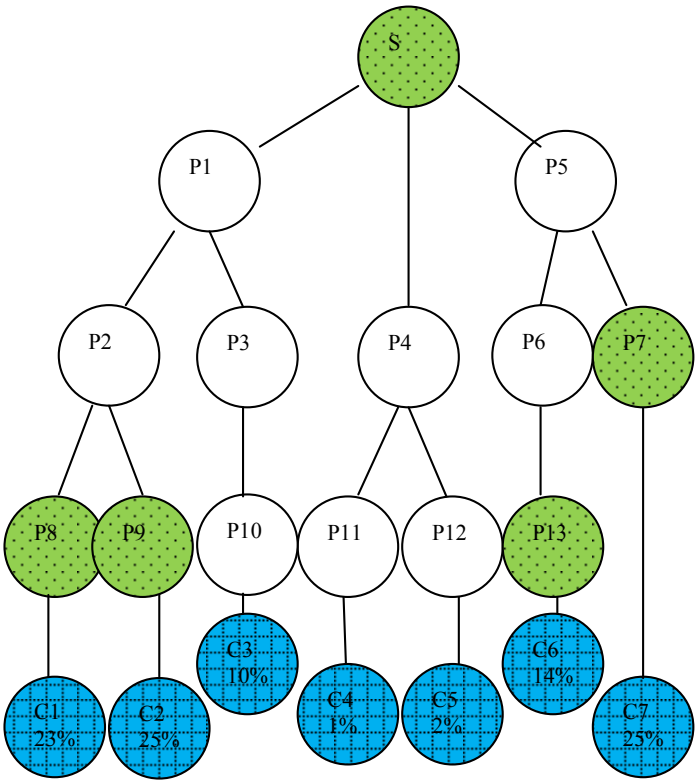


Fig.3. Cross over of Genetic Algorithm



S: master server
P: potential proxy site
C: client
%: request rate

Fig.4. result of genetic algorithm for choosing 4 proxies

V. Simulation Result for a Simple Topology

In our simulation, each path from client to master sever forms a linked list where each linked list node is a object which has two date members, one integer value of network node label and another Boolean variable represent if the network node is a proxy. After that insert the head pointer of each linked list to a new liked list whose head point called topology handle, through topology handle we can access each path (from client to master server) by traverse each linked list. Each population of genetic algorithm is an array of size "proxy number +1", the first element is the cost for that particular proxy placement, the rest elements of the array store the labels of the proxy placement. When each population is inserted into a linked list, these populations will be sorted by their cost. For the simple topology and client access pattern in Fig.4, we have 13 potential proxy sites, 7 client sites, request rate (normalized) are client 1: 23% , client 2: 25%, client 3 : 10%, client 4: 1%, client 5: 2%, client 6: 14%, client 7: 25%. The simulation results from genetic algorithm are in table 1. The results from greedy algorithm are in table 2. Compare these two tables the result is encouraging. When proxy number is 1 or 2, the results from the two algorithms are the same. In other cases (proxy numbers are 3, 4, 5) the results from genetic algorithm are out perform results from greedy algorithm(costs are less) .

Table1. Results from genetic algorithm

proxy number	cost	results
1	1.76	2
2	1.26	2, 7
3	0.78	7, 8, 9
4	0.36	7, 8, 9, 13
5	0.06	7, 8, 9, 10, 13

Table2. Results from greedy algorithm

proxy number	cost	result
1	1.76	2
2	1.26	2, 7
3	0.84	2, 7,13
4	0.54	2, 7, 10, 13
5	0.29	2, 7, 9, 10, 13

We tested a series of different request rate and plot the graph as Figure 7. The x axis is different sets of request rate, the y axis is costs for replica placements. The series1 is for results from greedy algorithm. Series2 is for results from genetic algorithm. In all request rate, greedy algorithm is better than greedy algorithm except for request rate 10. For request rate 10, the cost for replica placement from both algorithm are the same.

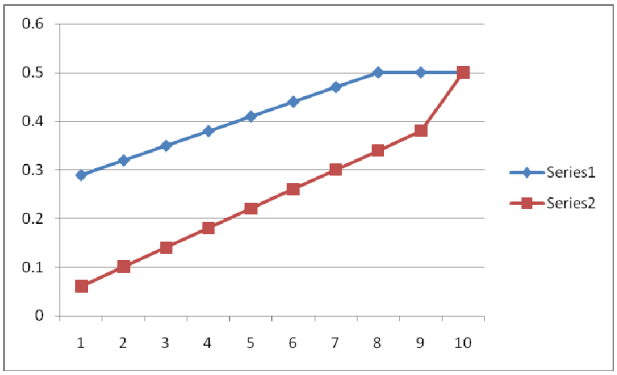


Fig 5 compare costs between greedy algorithm and genetic algorithm

VI. Future studying

Although the results from the simple topology are encouraging, we need to test more realistic network model to see if the genetic algorithm yield better results. Since in genetic algorithm, there is never a single "correct" setting for the parameters, we need to analyze the impact of the parameters. In particular we need experiment with different initial population, mutation rate, mutation step size, different methods of crossover, selection strategy and finally under what condition to terminate the process.

References

[1] Akamai. http://www.akamai.com
[2] Digital Island. http://www.digitalisland.com
[3] Exodus. http://www.exodus.com
[4] *Sivasubramanian, S., Szymaniak, M., Pierre, G., and Steen, M.,* Replication for web hosting systems.
ACM Computing surveys, Sep.2004: 291-334.
[5] Li, B., Golin, M. J., Italino,G. F., and Deng, X. 1999. On the optimal placement of web proxies in the internet. In *Proc. 18th INFOCOM Conference* (New York, N.Y.). IEEE Computer Society Press, Los Alamitos, CA. 1282–1290.
[6] Qiu, L., Padmanabhan, V., and Voelker, G. 2001. On the placement of web server replicas. In *Proc. 20th INFOCOM Conference* (Anchorage, AK). IEEE Computer Society Press, Los Alamitos, CA., 1587–1596.
[7] Radoslavov, P., Govindan, R., and Estrin, D. 2001. Topology-informed internet replica placement. In *Proc. 6th Web Caching Workshop* (Boston, MA). North-Holland, Amsterdam, The Netherlands.
[8] Rabinnovich, M. and Aggarwal, A. 1999. Radar: A scalable architecture for a global web hosting service. *Comput. Netw. 31*, 11–16, 1545–1561.
[9] Rodriguez, P. and Sibal, S. 2000. SPREAD: Scalable platform for reliable and efficient automated distribution. *Comput. Netw. 33*, 1–6, 33–46.
[10] Kangasharju, J., Roberts, J. and Ross, K. 2001a. Object replication strategies in content distribution networks. In *Proc. 6th Web Caching Workshop* (Boston, MA). North-Holland, Amsterdam, The Netherlands.
[11] Goldberg, D.E., *Genetic algorithms in searching, Optimization & machine learning,* Addison-Wesley, 1989.