Optimal Location of Web Servers

Girish Srinivasan and Natarajan Gautam
Department of Industrial Engineering
Penn State University
University Park, PA 16801

Abstract

The problem of optimally locating web servers across the Internet is considered. Assuming that the user demand forecast is known, the objective is to minimize cost, subject to satisfying performance requirements of customers (that access the web servers) in terms of throughput and latency thresholds. To solve the optimization problem, three heuristics are considered: greedy heuristic, DEJAVU algorithm and genetic algorithm. They are compared against each other for solution quality and speed.

Keywords

Facility location, Quality of Service, Web performance, heuristics

1. Introduction

The Internet is a key enabler of trade and commerce in the present day, keeping companies and its customers connected at all times. The growing reliance on connectivity to the World Wide Web (WWW), coupled with an exponential growth in Internet traffic over the years has led to increased rates of Internet congestion. As a result, users have begun to experience poor quality of service (QoS) as a direct consequence of increased network and server loads. QoS refers to the ability to provide a consistent and predictable data delivery service. In the Internet, this service can be expressed in terms of (i) ease of access to web sites, (ii) short response times from a web site, (iii) fast connections to the internet, and (iv) fast downloads.

When a company is unable to provide a sustained QoS level to its clients, it severely affects its customer service levels. In [12] it is stated that Zona Research, a market research and strategic consulting firm for the Internet industry, has calculated that nearly one out of two online buying attempts made by consumers on dial-up and mid-band connections are abandoned, equaling more than $25 billion in potential lost business due to poor Web performance. In an effort to reduce the detrimental effects of network congestion, companies should consider installing mirror sites. A mirror site is a web site that is a replica of an already existing site, used to reduce network traffic (hits on a server) or improve the availability of the original site. Mirror sites increase the speed with which files or web pages can be accessed, as users can download files more quickly from a server that is geographically closer to them. Large companies will look at installing several of these mirror sites at locations all over the world in an effort to enhance QoS levels to their customers. The optimal allocation of these mirror sites in terms of location and the number of mirror sites to be installed is a strategic decision that companies need to make in order to improve their QoS levels.

In this paper, the following question is addressed: given the demand forecast for the number of simultaneous users at M locations what is the optimal number of mirror sites to use and at which locations to install these mirror sites? Henceforth this problem will be referred to as the Optimal Allocation Problem (OAP). The objective of the OAP is to minimize the cost of installing mirror sites, subject to fulfilling the QoS requirements of users. The OAP is similar to the well-studied facility layout problem in operations research. This is a combinatorial optimization problem that can be solved using a variety of techniques. Facility layout problems have been solved using various methods. Zhang and Kim [11] use evolutionary algorithms to solve the facility layout problem for a two-row machine layout problem. In terms of location problems on the WWW, Li et al [4] considered a fixed number of proxy servers (M) that need to be placed at M locations out of a given set of N locations. Using a dynamic programming approach, Li et al [4] obtain an optimal $O(N^2M^3)$ algorithm for a tree topology of locations. However, this heuristic does not guarantee an optimal solution and is computationally slow. Based on these observations, the use of alternative heuristics in solving the OAP is explored. Deb et al [1] describe an elitist non-dominated sorting genetic algorithm (GA) for multi-objective optimization, which will be used in this research. The property of elitism enables the GA to retain the most optimal

solutions through all the generations of its evolution. This GA provides an optimal solution set that includes alternative feasible solutions.

Section 2 describes the optimization problem in detail and shows how hard it is to formulate and solve optimally. Then a Mathematical Programming Formulation of the optimization problem is described in Section 3. Section 4 presents three heuristic algorithms, namely, DEJAVU algorithm, greedy algorithm and genetic algorithm to solve the OAP. Example problems are solved using these two algorithms. In Section 5, the results from the different heuristic algorithms are compared and conclusions are drawn. Concluding remarks and plans for future works are mentioned in Section 6.

2. Optimal Allocation Problem

The objective of the OAP is to minimize the cost of installing mirror sites subject to satisfying QoS constraints that ensure that a certain performance level is delivered. Consider M locations on a rectangular grid of size $(A \times B)$ such that $M = AB$. Every cell in the grid is assumed to generate some demand. To formulate the mathematical program the following notations are required. Let N be the demand forecast matrix of the number of simultaneous clients (i.e. customers or users) at the M locations. Therefore N_{ij}, an element of the $(A \times B)$ matrix N, is the forecast for the maximum number of simultaneous clients at location (i, j), $i \in 1,\ldots,A$, $j \in 1,\ldots,B$. Let $\tilde{N} = \sum_{i=1}^{A}\sum_{j=1}^{B} N_{ij}$ be the total number of simultaneous clients at all the M locations together. We define the decision variable x_{ij} as a binary variable denoting the presence or absence of a server at location (i,j) as:

$$x_{ij} = \begin{cases} 1 & \text{if a web server is located at (i, j)} \\ 0 & \text{otherwise} \end{cases}.$$

The total number of web servers used will be $\sum_{i=1}^{A}\sum_{j=1}^{B} x_{ij}$ and the location of the web servers can be obtained from the $(A \times B)$ matrix $X = [x_{ij}]$.

It is important to make an assumption about which clients would be attached to which server. To this effect the quantity $d(i,j,k,l)$ is defined as the distance (not Euclidean) between locations (i,j) and (k,l) such that $d(i,j,k,l) = |i - k| + |j - l| + 1$. It is assumed that each client attaches itself to its closest web server (in terms of distance defined in the equation above). If there is a tie for the closest server, the clients distribute equally (truncated above if necessary) among the closest servers. Define matrix $L(N,X) = [L_{ij}(N,X)]$ as a function of the demand distribution N and the web server locations X such that,

$$L_{ij}(N, X) = \begin{cases} \ell & \text{if } x_{ij} = 1 \text{ and } \ell\,(>0) \text{ clients are associated with server at (i, j)} \\ 0 & \text{if } x_{ij} = 0 \end{cases}$$

Also define matrix $S(N,X) = [S_{ij}(N,X)]$ as a function of the demand distribution N and the web server locations X that denotes the distances from web servers to their respective farthest clients. Therefore,

$$S_{ij}(N, X) = \begin{cases} s & \text{if } x_{ij} = 1 \text{ and the distance of the farthest client from server at j is s} \\ 0 & \text{if } x_{ij} = 0 \end{cases}$$

To explain the notations, an example is illustrated below. Consider six locations (M=6) spread on a (2×3) grid (A = 2 and B = 3) with the demand forecast matrix $N = \begin{bmatrix} 15 & 6 & 10 \\ 10 & 5 & 6 \end{bmatrix}$. Consider a potential web server location matrix to be $X = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$. Then $L(N, X) = \begin{bmatrix} 15 & 9 & 16 \\ 13 & 0 & 0 \end{bmatrix}$ and $S(N, X) = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix}$. This example is illustrated in Figure 1.

A vector $\Psi_{ij} = [\psi_{ij}(\Delta)]$ is defined such that if Δ is the distance of the farthest client from the server at location (i,j) then $\psi_{ij}(\Delta)$ is the maximum number of clients that will be allowed to be attached to the web server such that certain QoS measures (in terms of delay, throughput, bandwidth and loss) are satisfied. The vector Ψ_{ij} can be obtained using standard QoS analysis based on queuing networks. In this paper we do not go into the detail of computing Ψ_{ij} but

assume that the vector Ψ_{ij} is supplied as an input to the problem. Since there cannot be a client at a distance of $A+B$ or more from any location (i,j), ψ_{ij} is a $[1\times(A+B-1)]$ vector. For the example configuration stated above (illustrated in Figure 1), if ψ_{ij} = [20 16 8 0] for all (i,j), then the server configuration X is "*feasible*".

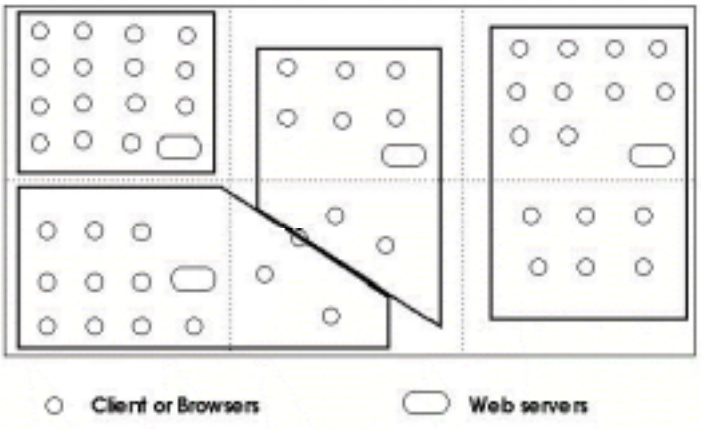

Figure 1 Illustration of N, X, $L(N,X)$ and $S(N,X)$

3. Mathematical Formulation

Let c_{ij} be the cost of allocating a server at any location (i,j). The mathematical programming problem is stated as:

$$\text{Minimize } Z = \sum_{i=1}^{A}\sum_{j=1}^{B} c_{ij}\,x_{ij}$$

Subject to:

$$L_{ij}(N,X) \le \Psi_{ij}(S_{ij}(N,X)) \quad \forall i = 1,...,A \text{ and } \forall j = 1,...,B \qquad \text{(i)}$$

$$\sum_{i=1}^{A}\sum_{j=1}^{B} x_{ij} \ge 1 \qquad \text{(ii)}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i = 1,...,A \text{ and } \forall j = 1,...,B$$

The constraint (i) ensures that the desired QoS level can be guaranteed. It is used to check if a server configuration can meet the QoS needs of the users. Constraint (ii) guarantees that the trivial solution $x_{ij} = 0 \forall i, j$ is not obtained.

The OAP can be formulated as an integer program. However, the formulation for even a small grid of the size 2×3 as an integer program with integer and binary variables and non-linear constraints results in a total of 78 (in general $AB + 2A^2B^2$) variables and 125 constraints (not including the 78 integer/binary constraints). The non-linearity needs to be removed using a standard linear transform for binary variables. However, this would increase the number of variables to 158 and the number of constraints to 547 constraints (not including the 158 integer/binary constraints). Even extremely powerful commercial integer programming packages cannot solve this simple problem to optimality in a reasonable time. Therefore it is out of the question to solve large problems by modeling them as integer programs. This prompts the need to develop heuristics and alternative methods that will provide acceptable solutions in reasonable time spans.

4. Methodology: Heuristics to Solve the OAP

One of the characteristics of the OAP is the relative ease to verify if a candidate solution X is feasible or not. This is taken advantage of in the three heuristics that are explained below. In terms of the results obtained, the key difference in the three heuristics are speed and accuracy of the solutions.

DEJAVU heuristic: In the heuristic DEJAVU (Decompose Evaluate Join Append Verify and Unplug), a large rectangular grid is decomposed into smaller partitions and then the partitions are regrouped to solve the large grid. The steps are as follows:

1. Decompose the A x B matrix into K partitions of sizes 3 x 2, 2 x 3, 2 x 4, 3 x 3 and 2 x 2
2. Evaluate the optimal server configuration within each of the K partitions by complete enumeration of all possible configurations.
3. Join the partitions to form the server configuration X and determine if X is feasible for the large regrouped grid.
4. Append the configuration with more servers by adding one server at each of the infeasible cells.
5. Verify if the new configuration is feasible. If it is feasible go to step 6 (with the new configuration), otherwise go to step 4.
6. Unplug unnecessary servers from the system (with the new configuration) by randomly removing servers and checking for feasibility.

Greedy Algorithm: The essence of this heuristic is to start with a mirror server in every cell of the grid and then one-by-one remove servers in a greedy fashion (starting with the cell with the largest cost to demand ratio and working its way to the smallest). A server is removed if and only if it results in a feasible server configuration. The following is the pseudo-code for the greedy algorithm with respect to the optimization problem. Let 'Feas' represent feasibility of a given server configuration. At the first iteration, the server configuration will be feasible because every cell is allocated a server initially. Hence, 'Feas' is initialized to 1, indicating feasibility of the initial server configuration.

1. Initialize server matrix X of size (A,B) such that each cell is allocated one server.
2. Calculate the Cost-Demand ratio for each cell i.e., c_{ij}/N_{ij}.
3. Let S be a sorted list of servers in the decreasing order of c_{ij}/N_{ij} $\forall i \in 1,...,A$ and $\forall j \in 1,...,B$.
4. Initialize R = $\varnothing$ and feas = 1.
5. For k = 1..AB
   {
   a. Choose element k in S, S_k (which is the highest c_{ij}/N_{ij} value), and undo server allocation at X(i,j).
   b. Ascertain feasibility of this new server configuration. Feas = 1 if feasible, else Feas = 0.
   i. If feas = 1,
   ii. $S = S - \{S_k\}$,
   iii. $R = R \cup \{S_k\}$, and
   iv. $feas = 1$.
   else
   i. Re-allocate server to X(*i*,*j*)
   }
6. R is the set of server locations in server matrix X that need to be allocated a server each.
7. End.

Genetic Algorithm: Here we look at a genetic algorithm (GA) that provides near-optimal results. The GA checks the feasibility of a randomly generated mirror server configuration and assigns a corresponding fitness value. A penalty is assigned each time the feasibility condition is violated. This process is repeated for each such configuration in a population of mirror server configurations. The fittest configurations are carried forward through successive generations, and subject to random mutations and crossovers. In the context of the OAP, mutation refers to randomly removing or assigning a server to a location, while crossover refers to the switching of a part (or parts) of a configuration between two individual configurations. GA parameters such as crossover and mutation probability and population size are varied to explore a larger solution space. With features like binary encoding, GAs lend themselves well to solving optimization problems like the OAP that deal with binary variables. Results from the GA described below are compared with those from the DEJAVU heuristic and the greedy algorithm, in Section 5.

The Non-dominated Sorting Genetic Algorithm (NSGA-II) [1] is a fast, elitist GA that has been used to solve the OAP. The NSGA – II varies from a simple genetic algorithm only in the way the selection operator works, while the crossover and mutation operators remain unchanged. The main properties of the NSGA – II Algorithm that warrant its use in solving the OAP are:

Non-dominated Sorting: This process ensures that non-dominated regions or Pareto-optimal fronts are searched, and results in quick convergence of the population towards non-dominated regions. This feature is useful while solving

multi-objective optimization problems, where no one solution may be optimal. Hence this will be useful in the proposed future work.
Elitism: NSGA – II provides for elitism, thus ensuring that the best solution achieved so far is carried forward to future generations. This facilitates large mutation probabilities, thereby increasing the solution search space. This is relevant in the present context, as the OAP deals with binary variables, and in order to have a large variety in the possible combinations of 0s and 1s, large mutation rates are used.

The NSGA – II algorithm generates a set of solutions that includes good, sub-optimal solutions before converging on one optimal value. Infeasible solutions are suppressed, although they are ranked and stored according to the level of constraint violation. Constraint violations are penalized in order to reflect a poor fitness value, resulting in a low ranking. The above features make the NSGA – II algorithm a good choice for solving the optimization problem. The NSGA – II is implemented using the *C* programming language. The code is modified include problem-specific variables. The objective function and its constraints are re-written in accordance with the mathematical programming formulation described in Section 3.

5. Results

In this paper, alternative methods to solve the Optimal Server Allocation Problem are suggested. The DEJAVU heuristic, a greedy algorithm and a genetic algorithm were developed and used to solve the OAP, and discussed in Section 4. In order to illustrate the efficacy of the adopted methodology in solving the optimization problem, the DEJAVU heuristic, the greedy algorithm and the genetic algorithm were tested on a large set of problems. Input parameters to the optimization problem were varied suitably to generate a variety of problems to test (i) the solution quality, and (ii) computation time of these three methods.

Solution (total cost in objective function)			Computation time		
DEJAVU heuristic	Greedy algorithm	GA	DEJAVU heuristic	Greedy algorithm	GA
56.48	58.28	49.65	6 s	2 s	3 min
18.65	14.61	12.6	2 s	2 s	4 min
53.44	51.81	43.27	5 s	2 s	6 min
30	27	25	12 s	3 s	8 min
65	60	55	8 s	4 s	12 min
155.44	133.65	133.65	34 s	18 s	1.5 hr
381.84	381.84	380.24	17 s	11 s	30 min
361.26	358.67	357.45	18 s	12 s	35 min
350	356	350	41 s	41 s	3.25hr
192	167	159	19 s	15 s	5 hr

Table 1 Comparative Solutions and Computation Times

From the results it is clear that the GA generates better solutions than the DEJAVU heuristic and the greedy algorithm, but is computationally slower. On the average, the greedy algorithm produced solutions that deviate from the GA solutions by 7 percent, and from the DEJAVU heuristic by 3 percent. On the average, the GA produced solutions that bettered the DEJAVU heuristic results by 10.5 percent and the greedy algorithm solutions by 7.1 percent. Computation times for a GA increase exponentially with population size. Computation times ranged from 30 minutes for problem size 5×7 to 3 – 4 hours for problem size 11×19 .

The effect of GA parameters - crossover probability, mutation probability, population size, the random seed used to initialize the population and the number of generations the GA is allowed to run, on the quality of the resulting solution is recorded. It is observed that choosing suitable values of the GA parameters can significantly reduce the computation time for a GA.

6. Conclusions and Future Work

The OAP is whether or not to allocate a mirror site to each cell in the mirror site grid. To solve this problem the DEJAVU heuristic, a greedy algorithm and a genetic algorithm are described in this paper. The following are the findings of the research work:

- The greedy algorithm and the DEJAVU heuristic are good options when computational speed is more crucial than solution quality.
- The GA gives near-optimal results and outperforms the DEJAVU heuristic and the greedy algorithm. It can be used as a benchmark in evaluating other heuristics. However, the GA is slow in converging to an optimal solution as it deals with a population of individuals. Another reason for the slow convergence is the fact that the GA evaluates completely random server configurations for feasibility, and takes time to "learn from past results". Computational speed of a GA can be improved by providing the GA with some initial feasible solution such as using the greedy heuristic's solution.
- GA parameters such as crossover probability, mutation probability and population size affect the GA's ability to find optimal solutions and hinder fast convergence to an optimal solution.
- Elitism in GAs facilitates large mutation probabilities thereby allowing for greater diversity in the solution alternatives. However, it can also cause the GA to get stuck at a local optimum for multiple generations, thereby slowing down the convergence process.

Future Work:

- Since the quality of the GA solution depends on GA parameters like crossover and mutation probability, an intelligent crossover or mutation scheme will be devised by which the crossover and mutation operations occur when the GA gets stuck on a local optimum.
- The OAP will be solved using different objective functions. In this paper, minimizing cost was the objective. This can be modified to minimizing the number of mirror sites that are allocated.
- The NSGA – II algorithm that was modified for use in this thesis lends itself well to solving multiobjective optimization problems. Hence, the OAP can be solved as a multiobjective optimization problem, where both cost and number of mirror site allocations can be minimized.
- The OAP can be applied to proxy-servers (web-servers that cache only a fraction of the parent server information) also. This will be addressed in the future.

References

1. K. Deb, S. Agarwal, A. Pratap and T. Meriyavan (2000). *"A Fast Elitist non-Dominated Genetic Algorithm for Multi-Objective Optimization: NSGA – II"*. In KanGAL Report No. 200001. Indian Institute of Technology, Kanpur, India.
2. C. M. Fonseca and Peter J. Fleming (1994). *"An Overview of Evolutionary Algorithms in Multiobjective Optimization"*. Evolutionary Computation.
3. M. J.Oates (2000). *"Evolutionary Algorithm Performance Profiles on the Adaptive Distributed Management Problem"*. BT Technology Journal.
4. B. Li, M. J. Golin, G. F. Italiano, X. Deng and K. Sohraby (1999). *"On the Optimal Placement of Web Proxies in the Internet"*. Proceedings of INFOCOM '99, pp. 1282-1290.
5. N. Srinivas and K. Deb (1994). *"Multiobjective Optimization Using Non-dominated Sorting in Genetic Algorithms"*. Department of Mechanical Engineering, IIT Kanpur, India.
6. John H. Holland (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press.
7. S. Glassman (1994). "A Caching relay for the World Wide Web". 1st Int. Conf. World Wide Web, Geneva, Switzerland.
8. V. Paxson and S. Floyd (1997). *Why we Don't Know How to Simulate the Internet*. In Proceedings of the 1997 Winter Simulation Conference, pp. 1037-1044.
9. M. Sayal, Y. Breitbart, P. Scheuermann and R. Vongralek (1998). *Selection Algorithms for Replicated Web Servers*. In Workshop on Internet Server Performance, Madison, WI.
10. David. S. Johnson and M. Pandya (1989). *"Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Traveling Salesman Problem"*. Annals of Operations Research, pp. 59-84.
11. Byoung – Tak Zhang and Jung – Jib Kim (2000). *"Comparison of Selection Methods for Evolutionary Optimization"*. Evolutionary Optimization.
12. http://www.applicationplanet.com/facts/facts_zona.html