ELSEVIER

European Journal of Operational Research 142 (2002) 396–418

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Computing, Artificial Intelligence and Information Technology

Performance analysis and optimization of web proxy servers and mirror sites

N. Gautam *

Department of Industrial and Manufacturing Engineering, The Pennsylvania State University, 310 Leonhard Building, University Park, PA 16802, USA

Received 22 December 1999; accepted 7 December 2001

Abstract

In this paper, web proxy servers and mirror sites that cache either partial or complete information of their parent web servers are considered. These proxy servers are usually located strategically, for example near high-user-demand locations. Using proxy servers would significantly reduce latency for the users in retrieving information as well as alleviate congestion in the network. The main problem addressed in this paper is determining the optimal number and locations of proxy servers in a network to minimize costs subject to delay, throughput and demand constraints. For a given set of proxy server locations, it is assumed that client or user requests at a location will always be sent to the nearest server. Thereby each client–server system can be modeled as an independent queueing network for which performance measures such as delay distribution and throughput are obtained. These performance measures are used in an optimization problem that is formulated to determine the optimal number and optimal location of proxy servers. A heuristic called the DEJAVU algorithm is developed to solve the optimization problem. Based on a comparison with a genetic algorithm, it can be concluded that the DEJAVU algorithm produces near-optimal to optimal results in a very short time.

Keywords: Internet; Web server; Queueing networks; Location; Performance modeling

1. Introduction

The proliferation of the Internet has resulted in a rapid increase in the number of Internet users and applications. This exponential growth (see [18]) has led to a large amount of congestion in the network. The World Wide Web (WWW) takes a lion's share of the Internet traffic. One way to alleviate the congestion due to WWW traffic, and also to reduce the latency in retrieving information from centralized web servers, is to use proxy servers or mirror sites for popular sites (see [6]). A proxy server stores some information

* Tel.: 1-814-865-1239; fax: 1-814-863-4745.
E-mail address: ngautam@psu.edu (N. Gautam).

0377-2217/02/$ - see front matter
PII: S0377-2217(02)00210-2

contained in its parent web server using caching strategies. A mirror server (also called replicated server) is a special type of proxy server that stores all the information of the parent web server. We shall refer to both servers generically as web servers.

These web servers need to be at strategic locations, based on user demand forecast. For example, there are 101 different replicated servers to access Netscape Browser, 15 servers for Yahoo in the US, 12 for Lycos, 8 for America Online, 7 for Alta Vista, 3 for Infoseek, etc. (see [19]). The users accessing information from a parent server can obtain that information from a nearby web server. For example, consider an electronic commerce application. Say a user sitting in London would like to shop by accessing a parent server in New York. If the New York web server used a web server near London then it would increase the speed of service which could potentially result in a higher probability of the user making a purchase! In fact, locating web servers strategically throughout the Internet can greatly improve customer service.

Li et al. [14] 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. [14] obtain an optimal $O(N^3M^2)$ algorithm for a tree topology of locations. In this paper, we consider not only obtaining the optimal location of proxy servers but also the optimal number of proxy servers. The objective function of the optimization problem is to minimize the cost of installation, operation and maintenance of proxy servers subject to satisfying constraints on demand as well as on performance measures such as throughput and delay. The problem is solved by first considering a single web server and modeling the client–server system (web server and browsers) as a queueing network to obtain required performance measures. Then the single-web-server model results are used in an optimization problem. The mathematical programming formulation is similar to the well-studied operations research problem of facility layout and location (see [4]).

In a recent paper, Wang et al. [24] addressed the facility layout problem with stochastic customer demand and immobile servers where customers travel to the closest server location (such as an automatic teller machine). Wang et al. [24] state that although there is related work in the facility location and allocation area (see [1,2,8,20]), they do not address the problem of locating servers in a service system in which the service locations are not pre-defined, servers cannot move or customers travel to the closest available server rather than along a pre-defined path. The authors look at Poisson arrivals with single visit to servers, the constraint is on expected waiting times, a Euclidean distance matrix is used, customers cannot be located at the facility node, and, there are m customer nodes and n candidate facility nodes. In this paper we consider a finite customer model with repeated visits to servers, the constraints are on delay distribution and throughput, a non-Euclidean distance matrix is used, customers and facilities can be on the same node, and there is an equal number of customer nodes and candidate facility nodes.

There is a vast amount of literature on queueing network models in computer and communication networks including [3,11–13,22,23,25]. However, one typically runs into a couple of difficulties while modeling systems as queueing networks: extremely large (multidimensional) state vectors and the inability to arrive at closed-form solutions for easy performance evaluation. In this paper the queueing networks are modeled as suitable continuous time Markov chains (CTMCs) such that the state vectors are relatively small and the results obtained are in terms of closed-form algebraic expressions.

In Section 2, web servers and browsers are modeled as simple client–server systems and CTMC analysis is used to obtain the required steady state probabilities. In Section 3, performance parameters such as throughput per client, probability distribution of delay between a client request and a server response, and server utilization are obtained. In Section 4 an optimization problem is formulated to determine the optimal number and location of web servers subject to satisfying demand, delay and throughput constraints. A heuristic called the DEJAVU algorithm is developed in Section 5. In Section 6, the DEJAVU algorithm is illustrated using an example. Then several experiments are conducted to compare the DEJAVU algorithm against a genetic algorithm. In Section 7 some generalizations to the client–server model developed in Section 2 are discussed. In Section 8 the conclusions are stated and directions for future work are examined.

2. Modeling web servers and browsers as client–server systems

The interaction between a client (more specifically, a browser) and a web server follows a request–response procedure where the client associates itself with a web server, makes periodic requests to the web server, and receives responses from the web server. There are several such clients or browsers associated with a web server at a given time. Each organization or company, owns or leases many web servers across the network. Fig. 1 illustrates an example of three web servers S_1, S_2 and S_3 to which respectively four, five and three clients are associated.

Consider an organization that is interested in setting up a few web servers at M potential locations across the Internet. This paper addresses the following question:

- Given the demand forecast for the number of simultaneous clients at each of M locations, what is the optimal number of web servers to use and where should they be installed?

There is a trade off in that installing more web servers would increase the cost whereas reducing the number of servers would worsen the Quality of Service (QoS) experienced by the browsers or clients. Also, with respect to the location, the cost is not necessarily uniform throughout the M locations, and the performance measures also vary depending on the location of the web servers. Two main QoS measures considered in this paper are: throughput and request–response delay for the clients or browsers. Therefore an optimal design should be a combination such that the cost of installation, operation and maintenance are minimized while requiring that each client or browser faces low request–response delay and high throughput.

For a given set of web server locations it is assumed that client requests will always be routed to the nearest web server. Thereby we can model each client–server sub-system independently and derive performance measures such as throughput, delay and server utilization. They will be used in Section 4 where we formulate and solve an optimization problem to address the main question of this paper.

2.1. A simplified representation of a single web server

Consider a single web server that processes requests from R browsers or clients. Each client sends a request to the web server and the web server queues up the client requests in a single queue and processes them according to a first come first served (FCFS) discipline. Assume that this web server has only a single

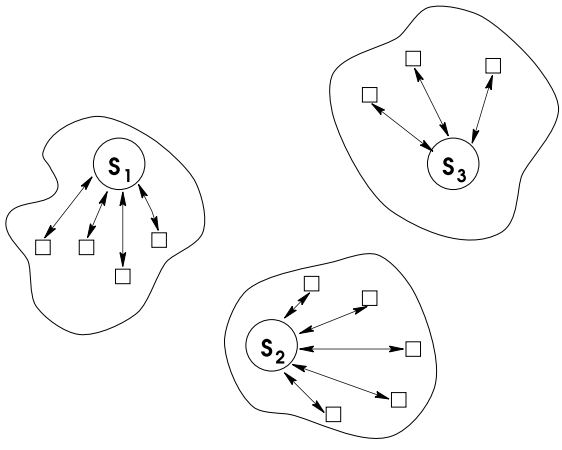

Fig. 1. Three identical web servers.

processor, therefore multitasking is not considered. Also assume that there are no losses due to buffer overflows.

Once the web server completes processing a client's request, it responds to the client by sending the processed information back to the client and then it starts processing the next request (if any). The client processes the piece of information that it received from the web server and when the processing is complete (this includes possibly some idle time at the client end), the client makes the next request to the web server (see Fig. 2). This is the finite population queue model [7]. Based on the current implementation of the HTTP protocol, we assume that all clients send only one request at a time and wait for a response (and processes the response) before sending out the next request (see [16,17]). Assume that the service times at the web server are i.i.d. exponential random variables with mean $1/\lambda$. The service times at the clients are assumed to be i.i.d. exponential random variables with mean $1/\mu$. Note that network delay times, propagation delay times, idle times before making requests (if any), etc. are also included in the client processing time.

2.2. Analysis

The system is analyzed by tagging a single client's requests and responses. This technique will prove to be useful while extending the model (see Section 7). Let $X(t)$ be the position of the tagged customer (corresponding to the request or response of the tagged client) in the web server queue at time t. Thereby, $X(t) = 0$ implies that the customer is at the browser or client queue, $X(t) = 1$ implies that the customer is being served by the web server and $X(t) = i$ for $i > 1$ implies that the customer is $i - 1$st in the waiting line to be served by the web server. Let $Y(t)$ be the total number of customers (including the one in service) in the web server queue at time t. Therefore $X(t) \leqslant Y(t)$ for all t. The stochastic process $\{(X(t), Y(t)), t \geqslant 0\}$ is a Continuous Time Markov Chain (CTMC) with infinitesimal generator

$$\begin{aligned}
&q_{(0,j),(0,j-1)} = \lambda \quad \forall 0 < j < R,\\
&q_{(i,j),(i-1,j-1)} = \lambda \quad \forall 0 < i \leqslant j \leqslant R,\\
&q_{(0,j),(j+1,j+1)} = \mu \quad \forall 0 \leqslant j < R,\\
&q_{(0,j),(0,j+1)} = (R-j-1)\mu \quad \forall 0 \leqslant j < R-1,\\
&q_{(i,j),(i,j+1)} = (R-j)\mu \quad \forall 0 < i \leqslant j < R,\\
&q_{(i,j),(k,l)} = 0 \quad \text{otherwise.}
\end{aligned} \tag{1}$$

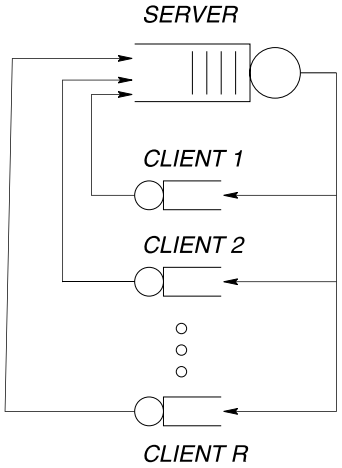

Fig. 2. R clients using a single server.

Using (1),

$$q_{(i,j),(i,j)} = -\sum_{(k,l)\neq(i,j)} q_{(i,j),(k,l)} = -(\min(j,1)\lambda + (R-j)\mu). \tag{2}$$

Let the long-run probability that the CTMC is in state (i,j) be $p_{(i,j)}$. Then solving the balance equations we have

$$\begin{aligned} p_{(0,j)} &= \frac{\left[\binom{R}{j} j!\lambda^{R-j}\mu^{j}\right]((R-j)/R)}{\sum_{k=0}^{R}\binom{R}{k}k!\lambda^{R-k}\mu^{k}} \quad \forall 0 \leqslant j < R, \\ p_{(i,j)} &= \frac{\left[\binom{R}{j} j!\lambda^{R-j}\mu^{j}\right](1/R)}{\sum_{k=0}^{R}\binom{R}{k}k!\lambda^{R-k}\mu^{k}} \quad \forall 0 < i \leqslant j \leqslant R. \end{aligned} \tag{3}$$

3. Performance criteria and scalability for a single web server

In this section, the system modeled in Section 2 is considered. The system is scaled up (by adding more clients) and the performance measures are evaluated.

3.1. Throughput per client

One of the performance criteria that a client application looks for is the number of requests it can send to the web server per unit time. Define the throughput per client, τ, as the expected number of responses the client processes per unit time. By conditioning on whether the customer is at the client queue or the web server queue, it can be shown that

$$\tau = \mu P\{\text{The customer is at the client queue}\} = \mu \sum_{j=0}^{R-1} p_{(0,j)}, \tag{4}$$

where $p_{(0,j)}$ is from Eq. (3).

Note that τ is also the expected number of service completions by the web server per unit time for a given client. Essentially if the *cycle time* is defined as the amount of time it takes for a customer to complete a client service and a web server service cycle, then τ is the inverse of the expected cycle time. In other words it is the expected number of times the given customer would cross a point in the network per unit time.

Intuitively, one would expect that the throughput per client, τ, to be a decreasing function of R, the number of clients. Fig. 3 shows the scalability of the system with respect to throughput per client via the plot of τ as a function of R, with $\lambda = 10$ and $\mu = 1$. The decreasing curve confirms with intuition.

3.2. Probability of delay

Another performance criterion of interest is the delay in receiving a response from the web server to a request made by the client. Essentially this is the waiting time in the web server queue for a given client request. Since the throughput per client is the inverse of the expected cycle time, the expected waiting time in the web server queue is the expected cycle time less the expected service time at the client and can be written as $1/\tau - 1/\mu$. The expected delay does not provide any new information beyond that of the throughput per client, and hence a better performance measure is considered.

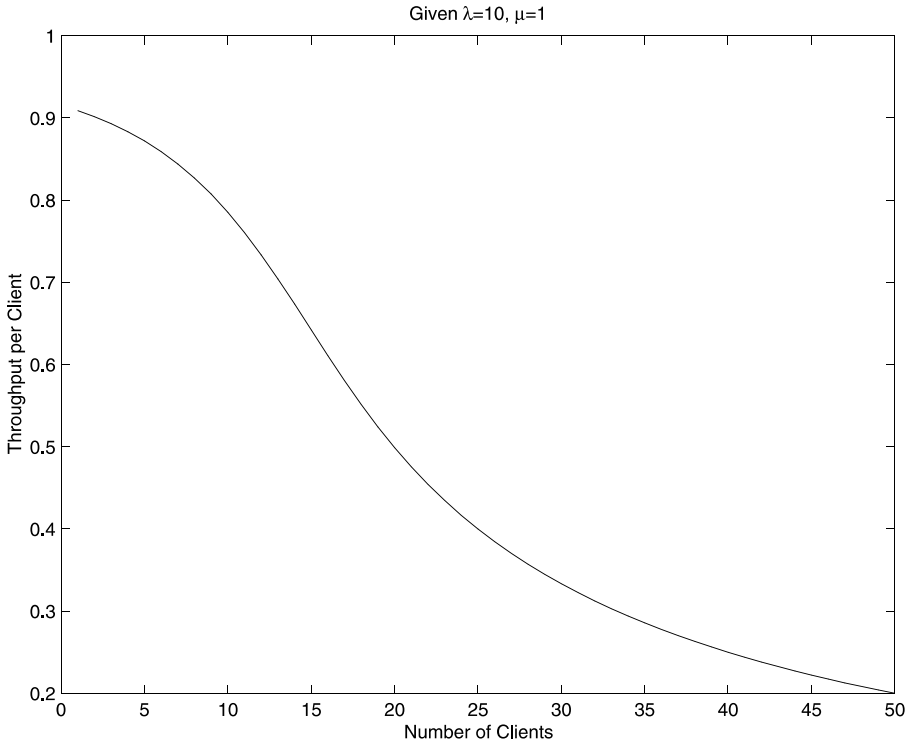

Fig. 3. Throughput per client.

Let π_δ be the probability that there is a larger-than-δ delay in receiving a response from the web server for a client request. In other words, this is the probability that an arriving client request spends more than δ amount of time at the web server queue (including the service). To calculate π_δ, we use Arrivals See Time Averages [26] or the Arrival Theorem [22] to obtain

$$P\{\text{the arriving customer to the web server queue will see } j \text{ customers in the queue}\} = p_{(0,j)}$$

for $j = 0, \ldots, R-1$. Let $Z(k)$ be an Erlang(k, λ) random variable with mean k/λ and variance k/λ^2. Let W be the waiting time of the arriving customer at the web server queue in the long run. Let J be the number of customers at the web server when the tagged customer arrives. Then

$$\begin{aligned}\pi_\delta = P\{W > \delta\} &= \sum_{j=0}^{R-1} P\{W > \delta \,|\, J = j\} P\{J = j\} = \sum_{j=0}^{R-1} P\{Z(j+1) > \delta\} \frac{p_{(0,j)}}{\sum_{i=0}^{R-1} p_{(0,i)}} \\ &= \sum_{j=0}^{R-1} \frac{p_{(0,j)}}{\sum_{i=0}^{R-1} p_{(0,i)}} \sum_{r=0}^{j} \mathrm{e}^{-\lambda\delta} \frac{(\lambda\delta)^r}{r!}, \end{aligned} \tag{5}$$

where $p_{(0,j)}$'s are from Eq. (3).

Since the delay would increase with the number of clients, one can expect π_δ to be an increasing function of R. See Fig. 4 for the scalability of the system with respect to the probability of delay for a given δ on the graph of π_δ versus R. The graph is consistent with the expectations.

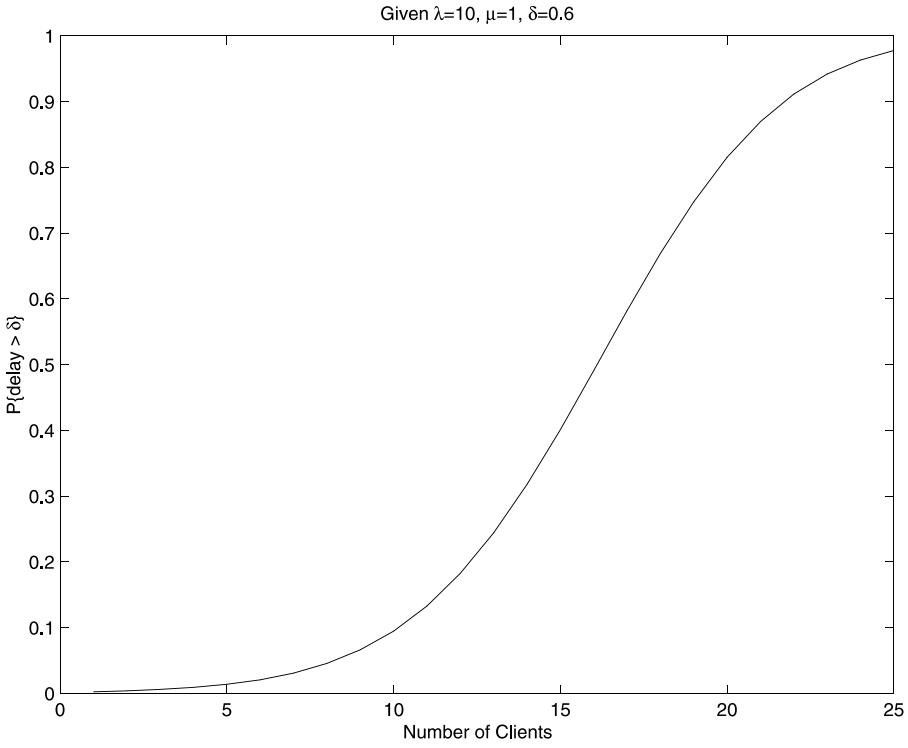

Fig. 4. Probability of delay.

3.3. Server utilization

While selecting an optimal number and optimal location of web servers, one of the criteria for cost optimization is server utilization. For the single web server system described in Section 2, define v as the web server utilization, which is the long-run probability that the web server is busy. Therefore

$$v = 1 - p_{(0,0)}, \tag{6}$$

where $p_{(0,0)}$ is from Eq. (3).

From a scalability point of view, the web server utilization should increase with the number of browser or clients. This intuition can be verified in Fig. 5.

4. Optimization problem

Consider an organization or company that would like to guarantee QoS in terms of throughput and delay for all its users accessing the company website. In order to do so, this organization would have to set up (based on the demand forecast) several web servers across the Internet. The optimization problem that we consider essentially determines whether or not to install a web server at each of the potential locations. We first explain the problem setting and then follow up with some useful notation before stating the optimization problem.

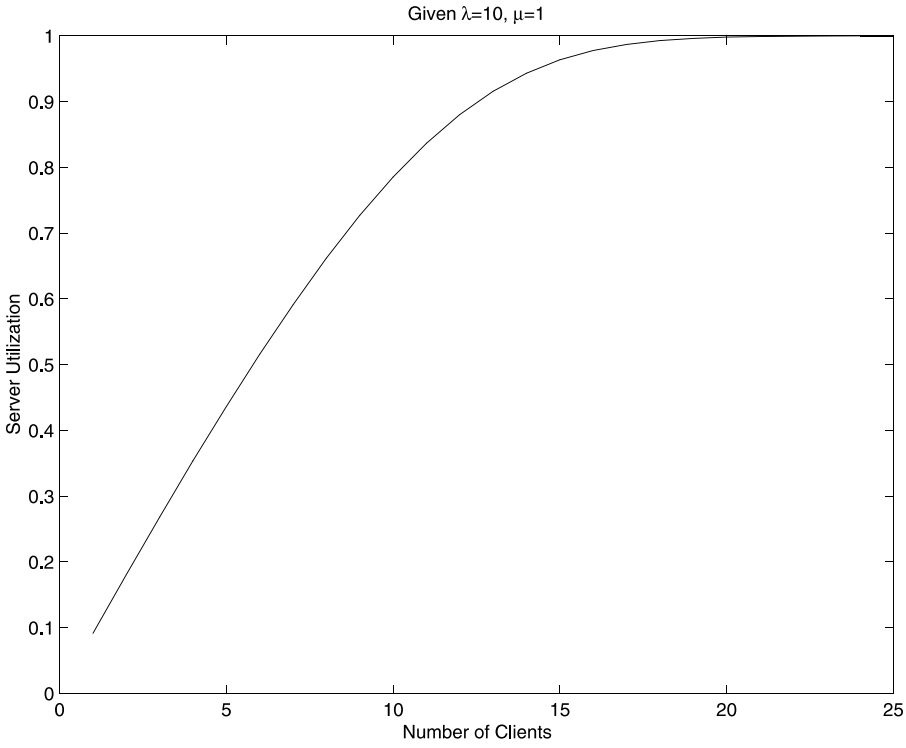

Fig. 5. Server utilization.

Problem setting:

1. There are M candidate locations where the web servers can be installed. Let the M locations be on an $A \times B$ rectangular grid such that $M = AB$. Based on the user demand forecast, the number of simultaneous browser requests from each cell in the grid can be obtained. Notice that there could be zero requests from a cell.
2. Due to the routing protocols, the requests from a browser/client is routed to its nearest server. If there is a tie for the closest server, the clients are distributed equally (rounding up if necessary) among the closest servers. Since usually the organizations interested in installing the web servers do not own the underlying network, other routing strategies besides the nearest server routing cannot be implemented.
3. The distance (not Euclidean) between locations (i, j) and (k, l) is defined as

$$d(i, j, k, l) = |i - k| + |j - l| + 1. \tag{7}$$

   This distance metric is used to identify nearest web servers.
4. There can be at most one web server in each cell. Also, if all the cells have one web server, that configuration is guaranteed to be feasible with respect to the demand and QoS constraints. This can be achieved by making the cell size suitably small (thereby more cells in total).

4.1. Notation

Consider the location set $\mathscr{M}$ formed by 2-tuples, $\mathscr{M} = \{(i, j) : i \in 1, \ldots, A,\ j \in 1, \ldots, B\}$. Let N be the demand forecast matrix of the number of simultaneous clients at the M locations. Therefore N_{ij}, an element

of the $A \times B$ matrix N, is the forecast for the number of simultaneous clients at location $(i,j) \in \mathscr{M}$. Note that the notation defined in Section 3, such as τ, v and π_δ, will use the subscript "i,j" (τ_{ij}, v_{ij} and $\pi_{\delta,(ij)}$) in this section to denote location (i,j). Define the decision variable x_{ij} as a binary variable denoting the presence or absence of a web 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 $X = [x_{ij}]$. 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

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

Thus, if a server is located at (i,j), $L_{ij}(N,X)$ represents the total number of clients (from location (i,j) and its neighborhood) associated with this server. Also define the 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 (as defined in Eq. (7)) from web servers to their respective farthest clients. Therefore

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

To illustrate the notation, consider the following example. There are six locations ($M = 6$) on a 2×3 grid ($A = 2$ and $B = 3$) with the following demand forecast matrix:

$$N = \begin{bmatrix} 15 & 6 & 10 \\ 10 & 5 & 6 \end{bmatrix}.$$

Suppose a potential web server location matrix X is

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Then

$$L(N,X) = \begin{bmatrix} 15 & 9 & 16 \\ 13 & 0 & 0 \end{bmatrix} \quad \text{and} \quad S(N,X) = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix}.$$

The above example is illustrated in Fig. 6.

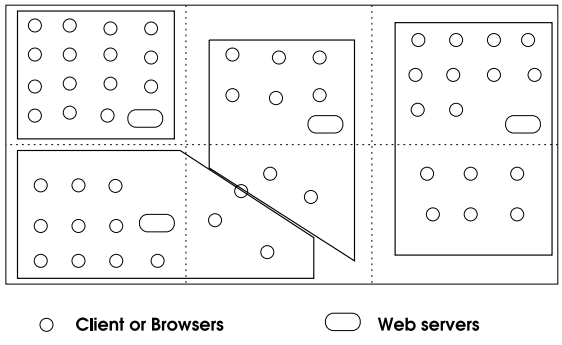

Fig. 6. Illustration of N, X, $L(N,X)$ and $S(N,X)$ for the 2×3 grid.

4.2. Mathematical programming model

Here, the mathematical programming formulation for the optimization problem is developed. Let c_{ij} and b_{ij} be respectively the set-up/maintenance cost per unit time and the operating cost per unit time for a web server at location (i,j). Then the average cost per unit time, Z, is given by

$$Z = \sum_{i=1}^{A} \sum_{j=1}^{B} c_{ij}x_{ij} + b_{ij}v_{ij}(N,X),$$

where $v_{ij}(N,X)$ is the average utilization (as a function of N and X) of the web server at location (i,j). The *key idea* is to decide at each location $(i,j) \in \mathscr{M}$, whether or not to install a web server, so that the average cost per unit time (Z) is minimized and the demand and QoS requirements (throughput and delay) are met. The QoS constraints for each client at $(i,j) \in \mathscr{M}$ are described as follows:

- the throughput as a function of N and X, $\tau_{ij}(N,X)$, must be at least as large as θ (this is the minimum throughput per client the web server would like to offer the clients),
- the probability that the client will face a delay larger than δ from the server (denoted by $\pi_{\delta,(ij)}(N,X)$) must be smaller than ϵ for each client (this is the maximum probability of a larger-than-δ delay the web server would like to offer the clients).

This can be stated as an optimization problem as

$$\begin{aligned}
\text{Minimize} \quad & Z = \sum_{i=1}^{A} \sum_{j=1}^{B} c_{ij}x_{ij} + b_{ij}v_{ij}(N,X) \\
\text{subject to:} \quad & \tau_{ij}(N,X) \geqslant \theta \quad \forall (i,j) \in \mathscr{M}, \\
& \pi_{\delta,(ij)}(N,X) \leqslant \epsilon \quad \forall (i,j) \in \mathscr{M}, \\
& x_{ij} = 0 \text{ or } 1 \quad \forall (i,j) \in \mathscr{M}.
\end{aligned}$$

Note that some of the equality constraints defining $v_{ij}(N,X)$, $\tau_{ij}(N,X)$, and, $\pi_{\delta,(ij)}(N,X)$ in terms of X and N have been left out of the formulation.

Next we formulate the optimization problem as a standard mathematical programming problem. It requires some pre-processing so that while solving the mathematical program, one can quickly check if the QoS constraints will be satisfied or not. In particular, we transform the QoS constraints so that they are in terms of $L(N,X)$ and $S(N,X)$.

4.2.1. Transforming the QoS constraints

See Section 2.1 for the definition of λ (the service rate or processing speed of the web server) and μ (the client processing rate that includes the network delay times, propagation delay times, processing times and idle times at the client). Now define $1/\lambda_{ij}$ as the mean processing time at web server in location (i,j) and $1/\mu_{kl,ij}$ as the mean processing time at client in location (k,l) that is attached to a server in location (i,j). Since the client processing time includes the network delay it is important to write it as a function of the distance from the web server. Hence define a function $f(\cdot)$ such that

$$\frac{1}{\mu_{kl,ij}} = f(d(i,j,k,l)). \tag{8}$$

If all the clients attached to a web server at location (i,j) are equidistant (where distance is defined as per Eq. (7)), then the client–server system at location (i,j) can be modeled as in Section 2.2 to obtain τ_{ij} and $\pi_{\delta,(ij)}$. However as a conservative approximation, for a server at location (i,j) calculate τ_{ij} assuming all clients are at the farthest location and $\pi_{\delta,(ij)}$ assuming all clients are at the nearest location (which will be at

distance 1 unless there is no demand at location (i,j)). This is based on the fact that τ and π_δ are decreasing functions of $1/\mu$. Refer back to Figs. 3 and 4 where $\lambda = 10$ and $\mu = 1$. If $\theta = 0.6$ and $\epsilon = 0.2$, then a maximum of about 13 clients and a maximum of about 12 clients can be accepted using the throughput and delay criteria respectively. Considering both criteria, a maximum of 12 clients can be accepted. In a similar manner it is possible to pre-compute for every location (i,j) a vector $\Psi_{ij} = [\psi_{ij}(\Delta)]$ such that if Δ is the distance of the farthest client from the web 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 the constraints on the throughput and delay can be satisfied. Note that there cannot be any client at a distance of $A+B$ or more from any web server. Therefore Ψ_{ij} is a vector of dimension $A+B-1$. An algorithm to obtain Ψ_{ij} is described below. It is called RHS since Ψ_{ij} forms the right-hand side term in the mathematical programming formulation in Section 4.2.2.

Algorithm RHS.
Input: A, B, λ, $f(\cdot)$
Output: $\Psi_{ij} = [\psi_{ij}(\cdot)]$ for all $(i,j) \in \mathscr{M}$
Method: For every $(i,j) \in \mathscr{M}$,
1. $d \leftarrow 1$
2. While $d \leqslant A+B-1$
   $\frac{1}{\mu} \leftarrow f(d)$
   Return $\psi_{ij}(d) \leftarrow \max\{n : \tau_{ij} < \theta \text{ and } \pi_{\delta,(ij)} < \epsilon\}$.

Given that it is possible pre-compute and store the Ψ_{ij} vector for each location (i,j), the following mathematical programming problem is formulated and algorithms to solve it are suggested where we assume that Ψ_{ij} is given for all $(i,j) \in \mathscr{M}$.

4.2.2. Mathematical programming formulation

Using the definitions from Section 4.1 for $L_{ij}(N,X)$ (i.e. the maximum number of clients associated with server in location (i,j) for a demand forecast N and a location matrix X) and $S_{ij}(N,X)$ (i.e. the distance of the farthest client from the server in location (i,j) for a demand forecast N and a location matrix X), the mathematical programming problem can be written as

$$\begin{aligned}
\underset{X}{\text{Minimize}} \quad & Z = \sum_{i=1}^{A}\sum_{j=1}^{B} c_{ij}x_{ij} + b_{ij}v_{ij}(N,X) \\
\text{subject to:} \quad & L_{ij}(N,X) \leqslant \psi_{ij}(S_{ij}(N,X)) \quad \forall (i,j) \in \mathscr{M}, \\
& \sum_{i=1}^{A}\sum_{j=1}^{B} x_{ij} \geqslant 1, \\
& x_{ij} = 0 \text{ or } 1 \quad \forall (i,j) \in \mathscr{M}.
\end{aligned}$$

Note that the constraint $\sum\sum x_{ij} \geqslant 1$ is used to avoid the trivial solution $x_{ij} = 0\ \forall i,j \in \mathscr{M}$. Another way to avoid the trivial solution is to define $L_{ij}(N,X) = \infty$ if $x_{ij} = 0$.

4.2.3. Example

To illustrate the constraint $L_{ij}(N,X) \leqslant \psi_{ij}(S_{ij}(N,X))$, consider the example in Section 4.1 where

$$N = \begin{bmatrix} 15 & 6 & 10 \\ 10 & 5 & 6 \end{bmatrix}.$$

Take location $(i,j) = (1,3)$ where the demand forecast for the number of simultaneous connections $N_{13} = 10$. For the location $(1,3)$, let $\lambda_{13} = 20$, $\theta = 0.6$, $\epsilon = 0.5$ and $\delta = 0.6$. Also for all (i,j), let

$$\frac{1}{\mu_{kl,ij}} = f(d(i,j,k,l)) = \alpha_{ij,kl} + d(i,j,k,l)\beta(ij,kl).$$

In particular, $\alpha_{13,ij} = 0.8$ and $\beta_{13,ij} = 0.2$. Calculating the value of Ψ_{13} using algorithm RHS (Section 4.2.1), we get $\Psi_{13} = [16\ \ 15\ \ 12\ \ 0]$, where $\psi_{13}(3) = 12$ implies that no more than 12 clients can be associated with the web server in location $(1,3)$ if the farthest of the 12 (or fewer) clients is at a distance 3 from the web server.

A potential web server location matrix

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

will yield

$$L(N,X) = \begin{bmatrix} 15 & 9 & 16 \\ 13 & 0 & 0 \end{bmatrix} \quad \text{and} \quad S(N,X) = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix}.$$

In that case, the location matrix X will be infeasible because the web server at location $(1,3)$ can accept at most 15 clients $(\psi_{13}(2) = 15)$ when the farthest client is at a distance 2 from the web server, whereas $L_{13}(N,X) = 16$ and $S_{13}(N,X) = 2$.

However, the location matrix

$$Y = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

will yield

$$L(N,Y) = \begin{bmatrix} 15 & 8 & 10 \\ 12 & 0 & 8 \end{bmatrix} \quad \text{and} \quad S(N,Y) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \end{bmatrix}.$$

This location matrix Y will result in a feasible constraint $L_{13}(N,Y) \leqslant \psi_{13}(S_{13}(N,Y))$ since $L_{13}(N,Y) = 10$, $S_{13}(N,Y) = 1$ and $\psi_{13}(1) = 16$. However, $L_{ij}(N,Y) \leqslant \psi_{ij}(S_{ij}(N,Y))$ needs to be checked for all locations (i,j) to determine if the location matrix Y is feasible.

5. The DEJAVU algorithm

Having formulated the optimization problem as a mathematical program in Section 4.2.2, we now describe an algorithm to solve it. We first make an assumption that for larger values of A and B, the set up/maintenance cost dominates the operating cost (i.e. $\min_{i,j} c_{ij} \geqslant \sum_{k,l} b_{kl}$). This permits us to drop the $b_{ij}v_{ij}(N,X)$ term from the objective function, as shown in Appendix A. Even for small problems the explicit formulation in Section 4.2.2 as an Integer Program (IP) requires a large number of variables and constraints. For example, in order to state the objective function and constraints in a linear form for a small 2×3 grid problem, we require 158 variables and 547 constraints (not including the 158 integer/binary constraints). The IP formulation of that example is shown in [5]. Even extremely powerful commercial integer programming software packages cannot solve this simple problem to optimality in a reasonable time. Thus it is out of the question to solve larger problems by modeling them as IPs. We hence resort to heuristics.

We develop a heuristic DEJAVU (Decompose Evaluate Join Append Verify and Unplug) that takes advantage of the relative ease of verifying if a candidate solution is feasible or not. This methodology to solve the optimization problem is analogous to the "min-cut clustering" approach [9] to the Traveling Salesperson Problem. In the min-cut clustering approach, a large number of cities are decomposed into smaller clusters and the smaller problems are solved within a cluster before regrouping. Similarly, in DEJAVU, we decompose a large $A \times B$ grid into smaller partitions and then regroup the partitions to solve the large grid.

For the optimization problem, we are given the demand forecast matrix N, cost matrix $C = [c_{ij}]$, server processing speeds $\lambda = [\lambda_{ij}]$, client processing time functions $f(\cdot)$ defined in Eq. (8), minimum throughput for each client θ, and, the maximum probability ϵ that the delay for a response is larger than δ. Using that we obtain the web server configuration via the DEJAVU algorithm as follows:

The DEJAVU algorithm.

1. Decompose the $A \times B$ matrix into K partitions $[\Gamma(1), \Gamma(2), \ldots, \Gamma(K)]$ of sizes 3×3, 2×3, 2×4, 3×2 and 2×2 using algorithm DECOMPOSE (Section 5.1).
2. Evaluate the optimal server configuration within each of the K partitions by complete enumeration via algorithm EVALUATE (Section 5.2).
3. Join the partitions to form the server configuration X and determine if X is feasible for the large regrouped grid using algorithm FEASIBLE (Section 5.3).
4. Append the configuration with more servers by adding one server at each of the infeasible cells using algorithm APPEND (Section 5.4).
5. Verify if the new configuration is feasible using algorithm FEASIBLE (Section 5.3). If it is feasible go to step 6 (with the new configuration $\overline{X}$), otherwise go to step 4.
6. Unplug unnecessary servers from the system (with the new configuration $\overline{X}$) using algorithm GREEDY (Section 5.5) to obtain the final configuration $\tilde{X}$. The resulting objective function is $\sum C_{ij}\tilde{X}_{ij}$.

5.1. Algorithm DECOMPOSE

We develop an algorithm to decompose an $A \times B$ matrix into K partitions of sizes 3×3, 2×3, 2×4, 3×2 and 2×2. Note that K is unknown and is an output of the algorithm. It is required that $B > 5$. Let $\Gamma(k)$ be the kth partition (where $k \in [1, K]$) denoted by the 4-tuple $[f_r, t_r, f_c, t_c]$ such that $\Gamma(k)$ is from row f_r to row t_r, and from column f_c to column t_c of the original matrix.

Algorithm DECOMPOSE.

Input: A, B

Output: K and $\Gamma(k)$ for $k = 1, \ldots, K$

Method:

1. $a \leftarrow A \bmod 3$ and $b \leftarrow B \bmod 3$. $A_2 \leftarrow 0$, $B_4 \leftarrow 0$ and $k \leftarrow 0$.
   If $a = 0$, $A_3 \leftarrow a/3$.
   If $a = 1$, $A_3 \leftarrow \lfloor a/3 \rfloor - 1$, $A_2 \leftarrow 2$.
   If $a = 2$, $A_3 \leftarrow \lfloor a/3 \rfloor$, $A_2 \leftarrow 1$.
   If $b = 0$, $B_3 \leftarrow b/3$.
   If $b = 1$, $B_3 \leftarrow \lfloor b/3 \rfloor - 1$, $B_4 \leftarrow 1$.
   If $b = 2$, $B_3 \leftarrow \lfloor b/3 \rfloor - 2$, $B_4 \leftarrow 2$.

2. $u \leftarrow 1$ and $v \leftarrow 1$. While $u \leqslant A_3$
   While $v \leqslant B_3$
   $k \leftarrow k+1$, $\Gamma(k) \leftarrow [3(u-1)+1, 3u, 3(v-1)+1, 3v]$ and $v \leftarrow v+1$.
   $u \leftarrow u+1$.
3. If $A_2 > 0$ $u \leftarrow 1$ and $v \leftarrow 1$. While $u \leqslant A_2$
   While $v \leqslant B_3$
   $k \leftarrow k+1$, $\Gamma(k) \leftarrow [3A_3 + 2(u-1)+1, 3A_3 + 2u, 3(v-1)+1, 3v]$ and $v \leftarrow v+1$.
   $u \leftarrow u+1$.
4. If $B_4 > 0$ $u \leftarrow 1$ and $v \leftarrow 1$. While $u \leqslant A_3$
   While $v \leqslant 2B_4$
   $k \leftarrow k+1$, $\Gamma(k) \leftarrow [3(u-1)+1, 3u, 3B_3 + 2(v-1)+1, 3B_3 + 2v]$ and $v \leftarrow v+1$.
   $u \leftarrow u+1$.
5. If $A_2 > 0$ and $B_4 > 0$ $u \leftarrow 1$ and $v \leftarrow 1$. While $u \leqslant A_2$
   While $v \leqslant B_4$
   $k \leftarrow k+1$, $\Gamma(k) \leftarrow [3A_3 + 2(u-1)+1, 3A_3 + 2u, 3B_3 + 4(v-1)+1, 3B_3 + 4v]$ and
   $v \leftarrow v+1$.
   $u \leftarrow u+1$.
6. Return $K \leftarrow k$ and $\Gamma(k)$ for $k = 1, \ldots, K$

As an illustrative example, we decompose a 10×13 matrix into $K = 18$ partitions (Fig. 7).

5.2. Algorithm EVALUATE

One way of solving the optimization problem is to enumerate all possible combinations for X and pick the best solution among all the feasible ones. For each of the k partitions, the optimal allocation problem is solved by complete enumeration using the following algorithm:

Algorithm EVALUATE.
Input: C, N, $\Gamma(k)$, Ψ_{ij} for all $(i,j) \in \mathscr{M}$
Output: X^* and Z^*
Method:
1. $[f_r, t_r, f_c, t_c] \leftarrow \Gamma(k)$, $C \leftarrow C(f_r : t_r, f_c : t_c)$, $N \leftarrow N(f_r : t_r, f_c : t_c)$, $Z^* \leftarrow \sum C_{ij}$.
2. $\mathscr{U} \leftarrow \{U : U(i,j) = 0 \text{ or } 1 \ \forall i \in [1, t_r - f_r + 1], j \in [1, t_c - f_c + 1]\}$, the universal set of all possible web server combinations
3. For every $X \in \mathscr{U}$
   If X is feasible using algorithm FEASIBLE
   If $\sum C_{ij}X_{ij} \leqslant Z^*$, $Z^* \leftarrow \sum C_{ij}X_{ij}$ and $X^* \leftarrow X$.
4. Return X^* and Z^*

Now we present an example where the optimal number and location of servers are obtained by enumerating all possible values of X. Consider the example in Section 4.1 where

$$N = \begin{bmatrix} 15 & 6 & 10 \\ 10 & 5 & 6 \end{bmatrix}.$$

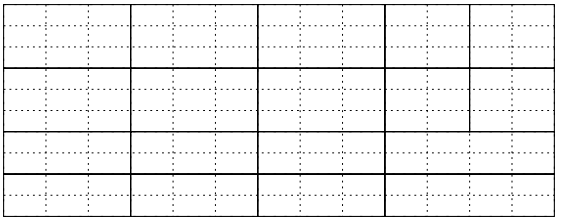

Fig. 7. Decomposing the 10×13 matrix into $K = 18$ partitions.

For the optimization problem, the following numerical values are used: $\lambda_{ij} = 20$ for all locations (i, j), $\theta = 0.6$, $\epsilon = 0.5$, and $\delta = 0.6$. To calculate μ, let $f(\cdot)$ defined in Eq. (8) be $f(d(i,j,k,l)) = \alpha_{ij,kl} + d(i,j,k,l)\beta_{ij,kl}$ (where $d(i,j,k,l)$ is defined in Eq. (7)) such that for all (k,l),

$$\alpha_{kl} = [\alpha_{ij,kl}] = \begin{bmatrix} 0.8 & 0.9 & 0.7 \\ 0.9 & 0.7 & 0.8 \end{bmatrix} \quad \text{and} \quad \beta_{kl} = [\beta_{ij,kl}] = \begin{bmatrix} 0.2 & 0.1 & 0.2 \\ 0.3 & 0.1 & 0.3 \end{bmatrix}.$$

Also the costs are given as

$$C = [c_{ij}] = \begin{bmatrix} 4 & 5 & 3 \\ 2 & 3 & 6 \end{bmatrix}.$$

The optimal solution via algorithm EVALUATE is

$$X^* = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

with a total optimal cost $Z^* = 8$.

5.3. Algorithm FEASIBLE

Not all web server configurations are feasible, where by feasibility we mean the ability to satisfy QoS constraints for the users. In fact there may be locations (i, j) where $L_{ij}(N,X) > \psi_{ij}(S_{ij}(N,X))$. Therefore it is important to verify if a candidate configuration X is feasible or not. We use the variable *feas* such that if $feas = 1$, the configuration is feasible, and if $feas = 0$, the configuration is infeasible.

Algorithm FEASIBLE.
Input: X, N, A, B, Ψ_{ij} $\forall(i,j)$
Output: *feas*
Method:
1. $feas \leftarrow 1$, $i \leftarrow 1$, $j \leftarrow 1$.
2. While $i \leqslant A$
   While $j \leqslant B$
      If $X_{ij} = 0$
         If $L_{ij}(N,X) > \psi_{ij}(S(N,X))$, $feas \leftarrow 0$.
      $j \leftarrow j + 1$
   $i \leftarrow i + 1$
3. Return *feas*.

5.4. Algorithm APPEND

This algorithm is used to determine which location without a web server needs to be appended with a web server. Here we consider each of the infeasible locations (i,j) and if there is a location in its neighborhood without a server but a high demand, then introduce a server there. In this manner, we add more web servers until a feasible solution is obtained.

Algorithm APPEND.
Input: X, N, A, B, Ψ_{ij} $\forall(i,j)$
Output: (k,l), the location where a server needs to be appended
Method:
1. $i \leftarrow 1$, $j \leftarrow 1$.
2. While $i \leqslant A$
   While $j \leqslant B$
   If $X_{ij} = 1$ and $L_{ij}(N,X) > \psi_{ij}(S(N,X))$,
   $(k,l) \leftarrow \arg\max_{(m,n)}\{N_{mn} : d(i,j,m,n) = S_{ij}(N,X) \text{ and } X_{mn} = 0\}$
   $j \leftarrow j+1$
   $i \leftarrow i+1$
3. Return (k,l).

5.5. Algorithm GREEDY

After appending the configuration with several servers, although the resulting system would be feasible, it may result in a system with too many servers. Therefore we develop a greedy algorithm to unplug some of the servers in such a manner that right through the unplugging phase, the resulting configuration is always feasible.

Algorithm GREEDY.
Input: X, C, N, A, B, Ψ_{ij} $\forall(i,j)$
Output: $\tilde{X}$, the configuration after unplugging servers
Method:
1. $r_{ij} \leftarrow$ rank of cell (i,j) when N_{kl}/C_{kl} is sorted in an ascending order over all (k,l).
2. $i \leftarrow 1$, $j \leftarrow 1$, $q \leftarrow 1$, $\tilde{X} \leftarrow X$.
3. While $q \leqslant AB$
   While $i \leqslant A$
   While $j \leqslant B$
   If $X_{ij} = 1$ and $r_{ij} = q$, $\tilde{X}_{ij} \leftarrow 0$
   If X is not feasible using algorithm FEASIBLE, $\tilde{X}_{ij} \leftarrow 1$.
   $j \leftarrow j+1$
   $i \leftarrow i+1$
   $q \leftarrow q+1$
4. Return $\tilde{X}$

6. Results

We first illustrate the steps of the DEJAVU algorithm by creating a randomized example. Later we compare the DEJAVU algorithm with a genetic algorithm to test the performance of the DEJAVU algorithm.

6.1. Algorithm illustration

In this section, a numerical problem is solved using algorithm DEJAVU to illustrate the steps of the algorithm. The demand forecast matrix N and the cost matrix $C = [c_{ij}]$ as well as their sizes A and B are chosen randomly from discrete uniform distributions. The rationale for choosing the parameters of the uniform distribution is ease of presentation and ease of verification. However there are no computational constraints on the choice of numerical values unless the numbers are exponentially large. Using the same rationale, for all (i,j) and (k,l), $((i,j) \in \mathscr{M}$ and $(k,l) \in \mathscr{M})$, let $\lambda_{ij} = 50$, $1/\mu_{kl,ij} = f(d(i,j,k,l)) = \alpha_{ij,kl} + d(i,j,k,l)\beta_{ij,kl}$ such that $\alpha_{ij,kl} = 0.48$, and, $\beta_{ij,kl} = 0.02$. Also, $\theta = 1.7$, $\epsilon = 0.1$ and $\delta = 0.16$. Therefore for all (i,j), Ψ_{ij} can be pre-computed using algorithm RHS (Section 4.2.1) as $\Psi_{ij} = [23\ 23\ 19\ 9\ 0\ 0\ \cdots\ 0]$.

The randomly generated values for A, B, C, and, N were obtained as: $A = 11$, $B = 22$,

$$C = \begin{bmatrix}
1 & 3 & 3 & 2 & 1 & 2 & 3 & 1 & 1 & 3 & 1 & 1 & 3 & 3 & 2 & 1 & 1 & 2 & 2 & 2 & 1 & 3 \\
2 & 1 & 1 & 2 & 3 & 3 & 1 & 3 & 1 & 1 & 1 & 3 & 2 & 2 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 3 \\
1 & 1 & 1 & 2 & 3 & 2 & 3 & 2 & 3 & 3 & 3 & 1 & 2 & 3 & 1 & 2 & 1 & 3 & 2 & 1 & 3 & 3 \\
3 & 1 & 2 & 3 & 2 & 2 & 3 & 3 & 3 & 1 & 3 & 3 & 3 & 1 & 3 & 3 & 2 & 2 & 3 & 2 & 3 & 1 \\
1 & 2 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 3 & 3 & 1 & 3 & 2 & 3 & 3 & 3 & 1 & 2 & 2 & 2 & 1 \\
1 & 2 & 2 & 2 & 3 & 1 & 3 & 1 & 1 & 3 & 2 & 3 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 \\
3 & 1 & 3 & 1 & 1 & 3 & 3 & 1 & 2 & 1 & 3 & 2 & 3 & 2 & 3 & 3 & 1 & 3 & 2 & 3 & 3 & 3 \\
3 & 2 & 2 & 3 & 3 & 3 & 3 & 2 & 1 & 1 & 2 & 2 & 3 & 3 & 2 & 1 & 1 & 3 & 3 & 1 & 3 & 3 \\
2 & 2 & 3 & 1 & 1 & 3 & 2 & 2 & 1 & 2 & 3 & 1 & 3 & 3 & 3 & 2 & 1 & 2 & 1 & 1 & 3 & 1 \\
1 & 2 & 2 & 1 & 3 & 2 & 2 & 1 & 1 & 1 & 1 & 3 & 3 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 3 & 2 \\
1 & 3 & 2 & 1 & 3 & 2 & 3 & 2 & 1 & 1 & 1 & 1 & 1 & 3 & 2 & 1 & 3 & 3 & 2 & 3 & 1 & 3
\end{bmatrix}$$

and

$$N = \begin{bmatrix}
1 & 8 & 7 & 14 & 14 & 2 & 4 & 10 & 5 & 4 & 12 & 11 & 2 & 8 & 15 & 10 & 3 & 3 & 11 & 6 & 1 & 1 \\
13 & 9 & 2 & 10 & 10 & 5 & 15 & 2 & 6 & 15 & 15 & 12 & 12 & 11 & 8 & 5 & 10 & 15 & 15 & 14 & 9 & 9 \\
9 & 5 & 15 & 5 & 15 & 4 & 10 & 13 & 14 & 9 & 2 & 13 & 4 & 4 & 2 & 6 & 15 & 14 & 3 & 6 & 14 & 4 \\
14 & 11 & 14 & 11 & 11 & 2 & 7 & 2 & 6 & 7 & 1 & 11 & 9 & 10 & 1 & 12 & 1 & 3 & 13 & 1 & 2 & 11 \\
7 & 10 & 7 & 12 & 5 & 5 & 4 & 3 & 5 & 5 & 11 & 2 & 7 & 2 & 7 & 13 & 6 & 4 & 8 & 1 & 11 & 1 \\
9 & 14 & 12 & 8 & 7 & 8 & 14 & 1 & 14 & 15 & 2 & 6 & 13 & 13 & 1 & 12 & 5 & 7 & 15 & 1 & 3 & 4 \\
8 & 5 & 12 & 1 & 15 & 13 & 12 & 8 & 11 & 12 & 2 & 5 & 14 & 15 & 13 & 4 & 13 & 1 & 7 & 4 & 4 & 6 \\
15 & 2 & 9 & 9 & 2 & 5 & 8 & 7 & 9 & 11 & 3 & 13 & 6 & 8 & 13 & 7 & 1 & 8 & 6 & 6 & 1 & 13 \\
14 & 1 & 14 & 15 & 1 & 12 & 9 & 14 & 14 & 14 & 10 & 11 & 15 & 11 & 1 & 9 & 8 & 2 & 11 & 7 & 4 & 1 \\
7 & 15 & 14 & 8 & 6 & 4 & 4 & 12 & 6 & 15 & 3 & 3 & 4 & 5 & 6 & 10 & 4 & 7 & 14 & 4 & 1 & 8 \\
11 & 6 & 2 & 14 & 6 & 15 & 13 & 11 & 11 & 7 & 10 & 11 & 2 & 14 & 10 & 9 & 13 & 5 & 1 & 7 & 4 & 14
\end{bmatrix}.$$

Now we go over the six steps of the DEJAVU algorithm to solve the optimization problem:

Step 1: The (11×22) N matrix is broken into $(K = 31)$ smaller partitions as shown in Fig. 8 using algorithm DECOMPOSE.

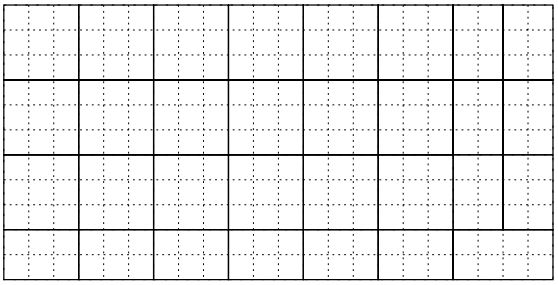

Fig. 8. Breaking down the given matrix N into cells.

Step 2: By complete enumeration using algorithm EVALUATE, the optimal server location subproblem in each partition is solved.

Step 3: The partitions are joined and the resulting X is

$$X = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}.$$

Using algorithm FEASIBLE we determine if X is feasible. It turns out that the above solution X is not feasible.

Steps 4 and 5: The following is the feasible solution $\overline{X}$ after incorporating additional web servers iteratively using algorithm APPEND and verifying if feasible using algorithm FEASIBLE (the additional web servers are indicated as $\widehat{\mathbf{1}}$)

$$\overline{X} = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \widehat{\mathbf{1}} & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & \widehat{\mathbf{1}} & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & \widehat{\mathbf{1}} & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \widehat{\mathbf{1}} & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \widehat{\mathbf{1}} & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & \widehat{\mathbf{1}} & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \widehat{\mathbf{1}} & 0 & 0 & \widehat{\mathbf{1}} \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & \widehat{\mathbf{1}} & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & \widehat{\mathbf{1}} & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & \widehat{\mathbf{1}} & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & \widehat{\mathbf{1}} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & \widehat{\mathbf{1}} & 0 & 0 & 0 & 1 & 0
\end{bmatrix}.$$

Step 6: Finally we run through algorithm GREEDY to unplug some of the web servers and result in the final solution $\tilde{X}$ as (the unplugged web server locations are indicated as $\widetilde{\mathbf{0}}$)

$$\tilde{X} = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & \widetilde{\mathbf{0}} & \widetilde{\mathbf{0}} & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & \widetilde{\mathbf{0}} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \widetilde{\mathbf{0}} & 0 & 0 & 0 \\
1 & 0 & 0 & \widetilde{\mathbf{0}} & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & \widetilde{\mathbf{0}} & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \widetilde{\mathbf{0}} & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & \widetilde{\mathbf{0}} & 0 & 1 & 0 & 1 & \widetilde{\mathbf{0}} & 0 & 1 & 0 & 0 & 1 & \widetilde{\mathbf{0}} & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & \widetilde{\mathbf{0}} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \widetilde{\mathbf{0}} & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & \widetilde{\mathbf{0}} & \widetilde{\mathbf{0}} & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}.$$

6.2. Comparisons

In this section we compare the performance of the DEJAVU algorithm against a genetic algorithm. These comparisons are made using 25 experiments with widely varying input parameters. In particular, we vary (i) the size of the location matrices from small to medium to large, (ii) the $A : B$ ratio from square to rectangular, (iii) the QoS constraints (ϵ, δ, θ), (iv) the web server and client processing parameters (λ) and (μ) from small to big, (v) the random variables for the cost matrix as discrete and continuous, (vi) the coefficient of variation of the random variables ranging from very small to very large for both C and N, and, (vii) the demand matrix N generated independently as well as inducing clusters of high/low values. Although we did not perform a full factor design of experiments study, we randomly selected 25 different combinations and ran experiments to demonstrate the performance of the DEJAVU algorithm.

A genetic algorithm was developed (details in [21]) to compare against the DEJAVU algorithm. The genetic algorithm code is derived from [10]. For each of the 25 experiments, we ran the genetic algorithm for several days under different initial conditions such as population size, number of generations, mutation probability, crossover probability and random seeds. Thereby we were able to obtain rigorously the optimal solution using the genetic algorithm. Note that the genetic algorithm must be solved over several days, while it takes only a few seconds to a minute using the DEJAVU algorithm. In Table 1 we demonstrate the quality of the DEJAVU algorithm solution by using the genetic algorithm as a benchmark. We report the percentage gap between the DEJAVU and the genetic algorithm solutions as

$$\%\text{gap} = \frac{\text{DEJAVU} - \text{genetic}}{\text{genetic}} \times 100.$$

The average gap is 3.68% over the 25 experiments. For 20 out of the 25 experiments, the gap is less than 5%. Also, for only 3 out of the 25 experiments, the gap exceeded 10%. Upon investigating the input parameters (defined in the early part of Section 6.2), it was found that there was no correlation between the algorithm performance and the input parameters. The main reason the DEJAVU heuristic performs poorly in some cases is because both the DECOMPOSE algorithm as well as the GREEDY algorithm result in making poor decisions. One way to improve the performance is to re-run the DEJAVU algorithm using the transpose matrices and take the better solution.

7. Generalizations

Emerging technologies geared toward improving the performance of web servers, clients and high-speed networks create the need for taking into account some generalizations to the model in Section 2.

Table 1
Comparing the DEJAVU solution against genetic algorithm

Index	DEJAVU solution	Genetic solution	% Gap
1	331.00	331.00	0.0000
2	32.00	32.00	0.0000
3	88.00	88.00	0.0000
4	50.38	49.65	1.4845
5	38.83	37.29	4.1324
6	83.20	78.65	5.7969
7	102.00	96.00	6.2500
8	69.55	69.55	0.0000
9	107.21	106.84	0.3547
10	376.92	376.93	0.0000
11	86.99	86.99	0.0069
12	357.63	357.45	0.0515
13	136.72	133.65	2.2970
14	62.55	61.18	2.2405
15	72.77	69.58	4.5798
16	48.27	43.27	11.5630
17	17.65	12.60	40.1127
18	34.00	34.00	0.0000
19	53.00	53.00	0.0000
20	64.00	64.00	0.0000
21	28.00	25.00	12.0000
22	103.00	103.00	0.0000
23	108.00	108.00	0.0000
24	161.00	159.00	1.2579
25	54.00	54.00	0.0000

Three potential generalizations are considered. In all three cases the system will be modeled as CTMCs. Once the CTMC modeling is complete it is a relatively straightforward exercise to write down the balance equations to obtain the steady-state probabilities. Using the steady-state probabilities, performance measures such as throughput, probability of delay and utilization can be obtained. These can be used in an optimization problem to obtain the optimal number and location of web servers. Therefore the CTMC models in the following generalizations are explained only briefly and the analysis, performance, optimization, etc. are left out as they can be done using the analysis followed in Sections 2–4.

7.1. Multi-processor server

Enhancement in the web server technologies could potentially result in multi-processor servers. In queueing theory terminology, at each web server, instead of using a single server queue as in Section 2, one could use a single queue with multiple servers (called multi-processors in this paper). If all the multiple processors are identical with mean processing time $1/\lambda$, then one could define a CTMC $\{(X(t), Y(t)), t \geqslant 0\}$, where $X(t)$ is the position of the given customer in the server queue and $Y(t)$ is the number of customers in the server queue at time t. If there are s processors, then $X(t) = 0$ implies the customer is at the client, $X(t) = 1, \ldots, s$ implies the customer is being processed, and $X(t) > s$ implies that the customer is the $X(t) - s$th customer waiting to be processed. The infinitesimal generator (Q) can be constructed and hence the performance parameters can be obtained.

7.2. Multi-class clients

The advent of differentiated services in the Internet, where the applications are differentiated into different classes of services, the performance measures will be based on the type of traffic. Also, in Section 4 a worst-case analysis is used to handle clients located at different distances from the server. It is hence possible to aggregate all similar clients (either based on the class of service or the distance from the server) and call them a single class of clients. Hence the network will have multiple classes of clients interacting with the server.

Assume that there are c classes of clients and a single processor server. For $j = 1, \ldots, R$, let $X_j(t)$ represent the class of the customer at the jth position in the server queue at time t, and, $X_j(t) = 0$ implies that there is no one at the jth position at time t. Clearly, $\{(X_1(t), X_2(t), \ldots, X_R(t)),\ t \geqslant 0\}$ is a CTMC. The evolution of the CTMC $\{(X_1(t), X_2(t), \ldots, X_R(t)),\ t \geqslant 0\}$ depends on the mechanism used to serve the different classes such as a priority mechanism. Based on the service mechanism, an appropriate generator matrix (Q) can be generated to obtain the required performance measures. However, the method scales very poorly with R. An alternative approach to obtaining the performance measures is through simulations.

7.3. Multi-request clients

The current protocol used at the browsers or clients allow only a single client request before a response is obtained from the server. This may soon be replaced by protocols that permit multiple client requests (see [15–17]). To model the scenario, assume that all the clients are identical and the server uses a single processor. Each client can send up to a maximum of m requests without waiting for a response from the server. Essentially this is similar to the model in Section 2 except instead of having one customer per client, there are m. Also two client buffers are needed in tandem, one for the replies to requests to be processed and the other to wait until a request needs to be sent.

The system is modeled using m tokens that circulate between each client and the server. Consider a single token of a single client. Let $X_s(t)$ be the position of the token in the server queue at time t, $X_c^1(t)$ be the position of the token in the first client queue at time t and $X_c^2(t)$ be the position of the token in the second client queue at time t. Let $Y_s(t)$, $Y_c^1(t)$ and $Y_c^2(t)$ be the number of tokens in the server queue, the first client queue and the second client queue respectively at time t. Then $\{(X_s(t), X_c^1(t), X_c^2(t), Y_s(t), Y_c^1(t), Y_c^2(t)),\ t \geqslant 0\}$ is a CTMC. Using the generator matrix Q, the performance measures can be obtained for this system.

8. Conclusions and future work

In this paper we considered the problem of optimally allocating web servers on a rectangular grid. The problem is formulated as a mathematical program with the objective of minimizing the cost of setting up and maintaining the web servers subject to satisfying demand and QoS constraints (in terms of throughput and probability of delay). We developed an algorithm called DEJAVU to solve the optimization problem. We compared the DEJAVU algorithm against a genetic algorithm. We showed that for 80% pf the problem instances, the gap between the DEJAVU and genetic algorithm was less than 5% and the average gap over all the problem instances was only 3.68%. Since the DEJAVU algorithm takes less than a minute to solve (as opposed to several days for the genetic algorithm), it would be a suitable option to solve the optimization problem.

This research paper provides a method for commercial organizations to decide the number and location of their proxy servers or mirror sites across the Internet to give its users high QoS. This paper also introduces to the operations research community a new and exciting problem to tackle.

In the future, improvising the DEJAVU algorithm and also obtaining other algorithms for optimality will be considered. In this paper a conservative maximum demand forecast for the clients is used and in future a demand distribution will be used. Also, the grid structure with physical distances would be replaced by the number of hops between the clients and server. Other objective functions as well as constraints will also be considered in the future. There is tremendous scope to improve upon this preliminary model.

Acknowledgements

The author would like to thank the anonymous reviewers for their comments and suggestions that led to considerable improvements in the content and presentation of this paper. The author also expresses gratitude to Dr. Hao Che for the initial discussions that led to the formulation of the main problem studied in this paper. The author is indebted to Mr. Hari Natarajan who provided valuable suggestions for the formulations and the algorithms used in this paper. The author is also very grateful to Mr. Girish Srinivasan for the genetic algorithm used in this paper.

Appendix A

Consider the optimization problem in Section 4.2.2. The following theorem states that if the set up/maintenance cost dominates the optimality cost (i.e. $\min_{i,j} c_{ij} \geqslant \sum_{k,l} b_{kl}$), the $b_{ij}v_{ij}(N,X)$ term can be dropped from the objective function.

Theorem A.1. *If* $\min_{i,j} c_{ij} \geqslant \sum_{k,l} b_{kl}$, *then it is never better to add a new web server and reduce the utilization of one or more of the existing web servers.*

Proof. Clearly for all (a,b), $0 \leqslant v_{ab}(N,X) \leqslant 1$. Consider the case for every (i,j) such that if $x_{ij} = 1$, $v_{ij}(N,X) = \gamma_{ij}$, (where $0 < \gamma_{ij} \leqslant 1$). Also, consider a location (k,l) such that $x_{kl} = v_{kl}(N,X) = 0$. Therefore one can write the objective function as

$$Z_1 = \sum_{(i,j):x_{ij}=1} c_{ij}x_{ij} + b_{ij}v_{ij}(N,X) = c_{kl} \times 0 + b_{kl} \times 0 + \sum_{(i,j):x_{ij}=1} [c_{ij}x_{ij} + b_{ij}\gamma_{ij}].$$

Now consider the case $x_{kl} = 1$ and $v_{kl}(N,X) > 0$ such that $v_{ij}(N,X) \leqslant \gamma_{ij}$ for every (i,j) such that $x_{ij} = 1$ and $(i,j) \neq (k,l)$. Therefore one can write

$$\begin{aligned}
Z_2 &= \sum_{(i,j):x_{ij}=1} c_{ij}x_{ij} + b_{ij}v_{ij}(N,X) \\
&= Z_1 + c_{kl}x_{kl} + \sum_{(i,j):x_{ij}=1} b_{ij}(v_{ij}(N,X) - \gamma_{ij}) \\
&\geqslant Z_1 + c_{kl}x_{kl} + \sum_{(i,j):x_{ij}=1} b_{ij}(0 - \gamma_{ij}) \\
&\geqslant Z_1 + c_{kl} \times 1 + \sum_{(i,j):x_{ij}=1} b_{ij}(0 - 1) \\
&\geqslant Z_1 + c_{kl} - \sum_{(i,j):x_{ij}=1} b_{ij} \\
&\geqslant Z_1.
\end{aligned}$$

Thus by increasing the number of web servers, the objective function increases (although the utilization decreases). Therefore to decide the optimal number of web servers it is enough if $\sum c_{ij}x_{ij}$ is used in the objective function by itself. To decide the location of the optimal number of web servers ties can be broken using the criterion to minimize $\sum b_{ij}v_{ij}(N,X)$. □

References

[1] O. Berman, R.C. Larson, S.S. Chiu, Optimal server location on a network operating as an $M/G/1$ queue, Operations Research 33 (1985) 746–771.
[2] O. Berman, R.C. Larson, N. Fouska, Optimal location of discretionary service facilities, Transportation Science 26 (1992) 201–211.
[3] D. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[4] R.L. Francis, L.F. McGinnis, J.A. White, Facility Layout and Location: An Analytical Approach, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[5] N. Gautam, An integer programming formulation of the web server location problem. Available from http://www.ie.psu.edu/people/faculty/gautam/gweb.pdf, 2001.
[6] S. Glassman, A caching relay for the World Wide Web, in: 1st International Conference on World Wide Web, Geneva, Switzerland, 1994.
[7] D. Gross, C.M. Harris, Fundamentals of Queueing Theory, third ed., Wiley, New York, 1998.
[8] M.J. Hodgson, A flow-capturing location–allocation model, Geographical Analysis 22 (1990) 270–279.
[9] E.L. Johnson, A. Mehrotra, G.L. Nemhauser, Min-cut clustering, Mathematical Programming, Series B 62 (1) (1993) 133–151.
[10] Kanpur Genetic Algorithms Laboratory, KanGAL. Available from http://www.iitk.ac.in/kangal/soft.htm.
[11] L. Kleinrock, Queueing Systems, Wiley, New York, 1975.
[12] V.G. Kulkarni, in: Modeling and Analysis of Stochastic Systems, Texts in Statistical Science Series, Chapman & Hall, London, 1995.
[13] P.R. Kumar, A tutorial on some new methods for performance evaluation of queueing networks, in: Advances in the Fundamentals of Networking – Part I: Bridging Fundamental Theory and Networking (special issue), IEEE Journal on Selected Areas in Communications, vol. 13, 1995, pp. 970–980.
[14] B. Li, M.J. Golin, G.F. Italiano, X. Deng, K. Sohraby, On the optimal placement of web proxies in the Internet, Proceedings of INFOCOM'99 (1999) 1282–1290.
[15] B.A. Mah, An empirical model of HTTP network traffic, in: Proceedings of INFOCOM'97, Kobe, Japan, 1997, pp. 592–600.
[16] J. Mogul, The case for persistent-connection HTTP, in: Proceedings of ACM SIGCOMM'95, Cambridge, MA, 1995, pp. 299–313.
[17] V. Padmanabhan, J. Mogul, Improving HTTP latency, in: Electronics Proceedings of the 2nd World Wide Web Conference on Mosaic and Web, Chicago, IL, 1994.
[18] V. Paxson, S. Floyd, Why we don't know how to simulate the Internet, in: Proceedings of the 1997 Winter Simulation Conference, 1997, pp. 1037–1044.
[19] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vongralek, Selection algorithms for replicated web servers, in: Workshop on Internet Server Performance, Madison, WI, 1998.
[20] J.G. Shanthikumar, D.D. Yao, Optimal server allocation in a system of multi-server stations, Management Science 33 (1987) 1173–1191.
[21] G. Srinivasan, Optimal allocation algorithms for web servers to guarantee QoS, M.S. Thesis, Industrial Engineering, Penn State University, 2001.
[22] J. Walrand, An Introduction to Queueing Networks, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[23] J. Walrand, P.P. Varaiya, High-performance Communication Networks, Morgan Kaufmann, Los Altos, CA, 1996.
[24] Q. Wang, R. Batta, C.M. Rump, A new class of facility location models with congestion, Naval Research Logistics (under review), 2001. Downloadable pdf version: http://www.acsu.buffalo.edu/~crump/pubs/immobile2.pdf.
[25] W. Whitt, Performance of the queueing network analyzer, The Bell System Technical Journal 62 (1983) 2817–2843.
[26] R.W. Wolff, Stochastic Modeling and the Theory of Queues, Prentice-Hall, Englewood Cliffs, NJ, 1989.