High-Density Model for Server Allocation and Placement *

Craig W. Cameron
Department of EE
Caltech, Pasadena, CA
cwcam@caltech.edu

Steven H. Low
Departments of CS and EE
Caltech, Pasadena, CA
slow@caltech.edu

David X. Wei
Department of CS
Caltech, Pasadena, CA
weixl@cs.caltech.edu

ABSTRACT

It is well known that optimal server placement is NP-hard. We present an approximate model for the case when both clients and servers are dense, and propose a simple server allocation and placement algorithm based on high-rate vector quantization theory. The key idea is to regard the location of a request as a random variable with probability density that is proportional to the demand at that location, and the problem of server placement as source coding, i.e., to optimally map a source value (request location) to a codeword (server location) to minimize distortion (network cost). This view has led to a joint server allocation and placement algorithm that has a time-complexity that is linear in the number of clients. Simulations are presented to illustrate its performance.

Categories and Subject Descriptors

C.2.5 [**Computer Systems Organization**]: Computer-communication Networks—*Local and Wide-Area Networks*; C.4 [**Computer Systems Organization**]: Performance of System

General Terms

Performance; Theory; Algorithms

Keywords

Content distribution, server placement and allocation, high density

1. INTRODUCTION

A content distribution network (CDN) reduces propagation delay, relieves server load, balances network traffic, improves service reliability, and disperses flash crowds. Content from a provider is distributed to multiple servers in the network, and a client request is served by a 'nearest' server. Here, proximity may refer to geographical distance, hop count, network congestion, server load or a combination. A central design issue is how to allocate and place servers in the network. In the context of peer-to-peer network or ad hoc wireless network, every node can be both a client and a server and the problem is to decide, in a distributed manner, how many copies of a file to store and where.

Large scale cooperative Web caches that have been deployed, e.g., the IRCache project [28] and the Harvest project [4, 6], typically adopt a hierarchical architecture, where a request is sent successively up a fixed hierarchy until the requested document is found. Cooperative Web caching have also been proposed that are hash-based [24, 14], directory-based [10, 19], and multicast-based [20, 27]. Most of these proposals assume a static configuration of the caching network. Interesting self-organizing strategies of wide-area cooperative caches is considered in [3, 23]. The proposal in [3] is to deploy a large number of smaller caches and store popular objects throughout this caching network evenly. A radius r is defined for each popular object. As this object is delivered, it is cached at caches in the path that are $r, 2r, 3r, \ldots$, away from the origin server. Hence, in steady state, it is populated in concentric rings in the network that are centered at the origin server with radii $r, 2r, 3r, \ldots$. This is enhanced by having caches communicate with neighbors about their holdings, so that a local miss can be directed to an appropriate neighbor. This can be done using the techniques in [10, 19]. The idea in [23] is to dynamically replicate servers based on current demand. A new server is added without content and the system will automatically decide which objects to migrate to it. The number and placement of servers and server selection are based on request pattern, client-server proximity, and server and network load.

The problem of optimal server allocation and placement is to decide how many servers to employ for each website and where to place them. Our ultimate goal is to develop simple distributed algorithms that can be used to self-organize CDN on a large scale dynamically based on current network traffic and client demands.

Server placement is known as the K-median problem in graph theory: given a graph with N nodes, each node i with a request rate $r(i)$, pick $K(< N)$ nodes as servers and assign each node to one of these servers so that the total weighted distance between all nodes i and their servers, weighted by $r(i)$, is minimized. This problem is shown in [15, 21] to be *NP*-hard for general graphs. Subsequent efforts have been to find polynomial algorithms to solve special cases and to find approximation algorithms to solve the general case. For tree graphs, [15] presents an $O(N^2K^2)$ algorithm to solve it optimally. This is improved to an $O(N^2K)$ algorithm in [26]. Dynamic programming based algorithms are also described in [18] (with complexity of $O(N^3K^2)$) and in [17] (with complexity of $O(NHK)$ where H is the height of the tree). An $O(\log N)$ *adaptive* algorithm to place one server in a general tree and an

*To appear in Proceedings of ACM Sigmetrics, Marina del Rey, CA, June 2002.

$O(\log^3 N)$ *adaptive* algorithm to place two servers in a complete binary tree are presented in [1]. Approximation algorithms for the general K-median problem has been studied in [2, 7, 8, 12, 9]. The best published approximation factor to date is 4 (i.e., the cost is at most 4 times the minimum), with running time $O(N^3)$, achieved in [9], based on the result of [12]. These algorithms, however, are not applicable to large scale self-organizing CDN we envision, for two reasons. First, the best approximation algorithms are based on primal-dual schema and Lagrangian relaxation, which have high running time for large N and more importantly, require global information throughout the execution. Second, these algorithms are concerned with the placement of K servers to serve a single website (content provider). In CDN, websites have greatly varying popularity [5] and it is crucial to exploit this diversity in server allocation.

In this paper, we take a completely different approach, focusing on the case where both client and server densities are high. In this regime, server placement can be regarded as a high-rate vector quantization problem. *The key idea is to regard the location of a request as a random variable with a probability density that is proportional to the demand at that location, and the problem of server placement as source coding, i.e., to optimally map a source value (request location) to a codeword (server location) to minimize distortion (network cost).* This view has led to a simple *joint* server allocation and placement algorithm with time complexity linear in NM where N is the number of clients (e.g., client side proxies) and M is the number of content providers; in particular, it is linear in N. Preliminary simulation results suggest that it has a good performance-complexity tradeoff.

In the rest of this paper, we will present the high-density model (Section 2) and derive a joint server allocation and placement algorithm (Section 3). We compare the performance of this algorithm with the best published approximation algorithm for the K-median problem through simulations (Section 4). Finally we conclude with limitations of this work (Section 5).

2. HIGH DENSITY MODEL

The proposal of [3] suggests the integration of storage with network where every router is potentially a small cache. Network nodes such as routers are in a best position to monitor traffic and self-organize into a large scale CDN that adapt their configuration to changes in client requests, network congestion, and server load. The main savings come from reduced network traffic and propagation delay, and the central issue is the optimal allocation and placement of these servers (or files in the context of peer-to-peer network or ad hoc wireless network). This problem can be viewed as a high-rate vector quantization with dimension two [11]. In this section, we explain how this view leads to a simple server allocation and placement algorithm, presented in the next section.

We start with the case of a single website, and extend it to the case of multiple websites. A 'website' in our model may represent a content provider, an entire website, a collection of files or applications, or a single file or application. A 'node' may represent an end user of the website, or more likely, a client-side proxy that serves a family of end users in the same local area network or same organization. By placing a server 'at a node', we mean placing a server 'near' the end user or client-side proxy represented by the node, e.g., on the same subnet.

2.1 Single website

Consider a set V of points (called *nodes*) in the $\Re^2$ plane indexed by $i = 1, 2, \ldots, I$. Let $z_i \in \Re^2$ be the coordinate of node i, and $Z = \{z_i \mid i \in V\}$ be the set of coordinates of all nodes; we will refer to a node both by i and by its coordinate z_i. Node i accesses the (single) website at a rate of $r(i)$ requests per minute. We are to place $K \geq 1$ servers at locations $s = (s_1, \ldots, s_K)$, where $s_k \in Z$, and the goal is to choose these locations s so as to minimize the total network cost of serving the requests, defined as follows.

Let $d(i, j)$ be the 'distance' of serving a request of node i by a server located at node j. Given server locations s, the distance measure $d(i, j)$ partitions V into what are called Voronoi cells $V_k \subseteq V$ defined by:

$$V_k \quad = \quad \{j \in V \mid d(j, s_k) \leq d(j, s_l), \ \ \forall l\}$$

such that $\cup_{k=1}^{K} V_k = V$. If $d(j, s_k) = d(j, s_l)$, ties are broken arbitrarily so that V_k are disjoint. Hence members of Voronoi cell V_k are *nearest neighbors* of server s_k.

The cost of serving a Voronoi cell V_k is $\sum_{j \in V_k} r(j) d(j, s_k)$. When there are K servers, the *network cost* is

$$c(K) \quad = \quad \min_{s} \sum_{k=1}^{K} \sum_{j \in V_k} r(j) d(j, s_k) \qquad (1)$$

where the minimization is over all server locations $s = (s_1, \ldots, s_K)$ in V.[1] This cost, in units of *rate-distance*, is a measure of minimum network capacity required, the minimum amount of network traffic, or total delayed weighted by demand, when there are K servers.

The case with Hamming distance measure is particularly simple, so we next present its solution as an illustration of various definitions.

Example 1: Hamming distance

Under the Hamming distance defined by

$$d(i, j) \quad = \quad \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

the network cost is

$$c(K) \quad = \quad \min_{s} \sum_{j \in V} r(j) \ \mathbf{1}(j \neq s_k \text{ for any } k = 1, \ldots, K)$$

where the indicator function $\mathbf{1}(A)$ is 1 if A is true and 0 otherwise. Hence the optimal locations are the first K nodes that have the largest request rates. This is called the *hot-spot* algorithm in [22]. Without loss of generality (relabeling nodes if necessary), we can assume that

$$r(1) \geq r(2) \geq \cdots \geq r(I)$$

The network cost under hot-spot algorithm is the total request rate, summed over the remaining $I - K$ nodes:

$$c(K) \quad = \quad \sum_{j=K+1}^{I} r(j)$$

■

[1] This formulation assumes that the location of the origin server (content provider) can also be optimized. This is optimistic but the effect of this approximation is insignificant for a large network.

Hamming distance assumes that the cost of serving a remote request is independent of how far the request is from the server, and hence is inappropriate. For proper accounting, the distance measure should take a larger value if the request is farther away from its nearest server. Euclidean distance is one such measure. As noted above, distance here may denote hop count, geographical distance, network congestion, server load or a combination. A major assumption in our model is that the metric adopted can be mapped (isometrically) into Euclidean distance for the application of interest.

We now approximate the network cost for the case when both nodes and servers are dense.

2.2 High-density approximation

Let *every* point $z = (z_1, z_2) \in \Re^2$ be a node. The request rate of node z is $r(z)$. Interpret the normalized request rate

$$f(z) \quad = \quad \rho^{-1} r(z) \quad \text{where } \rho := \int r(z)\, dz \tag{2}$$

as the *spatial density* of requests. The idea is to regard the location Z of a request as a random variable with probability density f, and the problem of server placement as source coding, i.e., to optimally map a source value (request location Z) to a codeword (server location s) to minimize distortion (network cost). We now apply the techniques of high-rate vector quantization [11, Chapter 5] to derive a server placement algorithm.

The network cost is defined in an analogous way to (1) as, for a single website and K servers,

$$c(K, s) \quad := \quad \rho \sum_{k=1}^{K} \int_{V_k(s)} f(z) d(z, s_k)\, dz$$

Here, $s = (s_1, \ldots, s_K)$ are the locations of the K servers, $V_k(s) = \{z | d(z, s_k) \leq d(z, s_l),\ \forall l\}$ is the Voronoi cell containing server location s_k. We assume that K is large so that $V_k(s)$ is small, and that $f(z)$ is smooth so that $f(z) \simeq f(s_k)$ over $V_k(s)$. We further assume that the distance function is the Euclidean distance, $d(z, s) = ||z - s||$. Then

$$c(K, s) \quad \simeq \quad \rho \sum_{k=1}^{K} f(s_k) \int_{V_k(s)} ||z - s_k||\, dz$$

Next we approximate the region $V_k(s)$ by a circular disk with the same area centered at s_k described by:

$$\{z \mid ||z - s_k|| \leq a_k\} \qquad \text{with } a_k := \sqrt{\frac{|V_k(s)|}{\pi}}$$

where $|A|$ denotes the area of set A. Then [2]

$$c(K, s) \quad \simeq \quad \rho \sum_{k=1}^{K} f(s_k) \int_{\{z : ||z - s_k|| \leq a_k\}} ||z - s_k||\, dz$$

By changing the variable of integration, we get

$$\begin{aligned} \int_{\{z : ||z - s_k|| \leq a_k\}} ||z - s_k||\, dz \quad &= \quad a_k^3 \int_{\{z : ||z|| \leq 1\}} ||z||\, dz \\ &= \quad \frac{2|V_k(s)|^{3/2}}{3\sqrt{\pi}} \end{aligned}$$

[2] Indeed, the following expression is a lower bound because, among sets of the same area, the moment of inertia about s_k of any set A is the smallest when it is a circular disk centered at s_k [11, Lemma 5.3.1].

and hence

$$c(K, s) \quad \simeq \quad \frac{2\rho}{3\sqrt{\pi}} \sum_{k=1}^{K} f(s_k) |V_k(s)|^{3/2} \tag{3}$$

To interpret, note that $f(s_k)|V_k(s)|$ is roughly the probability that a request falls in Voronoi cell $V_k(s)$. Then, (3) approximates the cost of serving *every* request in Voronoi cell $V_k(s)$ by $\frac{2}{3\sqrt{\pi}} \sqrt{|V_k(s)|}$, i.e., proportional to the square root of the cell's area. Hence, roughly, the sum

$$\frac{2}{3\sqrt{\pi}} \sum_{k=1}^{K} f(s_k) |V_k(s)|^{3/2}$$

is the expected cost of serving a request in the network, and the network cost $C(K, s)$ is roughly this per-request cost multiplied by the total request volume ρ.

We specify server location in this continuum model by *server density* $\lambda(z)$, with the interpretation that the *fraction* of servers in an infinitesimally small area dz around z is $\lambda(z)dz$. Hence the *number* of servers in any region A is $K \cdot \int_A \lambda(z)dz$. Note that $\int \lambda(z)dz = 1$ so λ can also be regarded as the probability density of server location. We approximate $\lambda(z)$ by $\lambda(s_k)$ over the small Voronoi cells $V_k(s)$. Hence, since there is exactly one server in each Voronoi cell, we have

$$1 \quad = \quad K \cdot \int_{V_k(s)} \lambda(z)\, dz \quad \simeq \quad K \cdot \lambda(s_k)\, |V_k(s)|$$

or

$$\sqrt{|V_k(s)|} \quad \simeq \quad \frac{1}{\sqrt{\lambda(s_k) K}}$$

Substituting into (3), and when K is large so that $V_k(s)$ is small, we have

$$\begin{aligned} c(K, s) \quad &\simeq \quad \frac{2\rho}{3\sqrt{\pi K}} \sum_{k=1}^{K} \frac{f(s_k)}{\sqrt{\lambda(s_k)}}\, |V_k(s)| \\ &\simeq \quad \frac{2\rho}{3\sqrt{\pi K}} \int \frac{f(z)}{\sqrt{\lambda(z)}}\, dz \end{aligned} \tag{4}$$

Given request density f, our goal is to choose a server density λ that minimizes the above approximate cost.

To simplify notation, define for any p-integrable function g, $0 < p \leq \infty$, its L_p norm as

$$||g||_p \quad = \quad \left(\int g(z)^p\, dz \right)^{1/p} \tag{5}$$

Using Hölder's inequality, it can be shown that the right-hand side (4) is lower bounded by [11, Chapter 5]

$$c^*(K) \quad = \quad \frac{2}{3\sqrt{\pi}}\, ||f||_{2/3} \cdot \frac{\rho}{\sqrt{K}} \tag{6}$$

and this lower bound is achieved with the server density

$$\lambda^*(z) \quad = \quad \left(\frac{f(z)}{||f||_{2/3}} \right)^{2/3} \tag{7}$$

where, from (5), $||f||_{2/3} = \left(\int f(z)^{2/3}\, dz \right)^{3/2}$. Using (2), we can also express $c^*(K)$ and $\lambda^*(K)$ directly in terms of the request rate

$r(z)$. We have $\rho||f||_{2/3} = \rho\left(\int f(z)^{2/3}\,dz\right)^{3/2} = \left(\int(\rho f(z))^{2/3}\,dz\right)^{3/2} = ||r||_{2/3}$. Hence $c^*(K)$ in (6) can be rewritten as

$$c^*(K) \;=\; \frac{2}{3\sqrt{\pi}} \cdot \frac{||r||_{2/3}}{\sqrt{K}} \tag{8}$$

and $\lambda^*(z)$ in (7) can be rewritten as

$$\lambda^*(z) \;=\; \left(\frac{r(z)}{||r||_{2/3}}\right)^{2/3} \tag{9}$$

For ease of reference, we will called λ^* the optimal server density and $c^*(K)$ the optimal cost in this paper.

Remarks:

1. This suggests a server placement strategy, when cost is measured by the Euclidean distance, where server density $\lambda^*(z)$ is proportional to the $2/3$-power of the request density, $f(z)^{2/3}$, or equivalently, of the request rate, $r(z)^{2/3}$. The strategy incurs an optimal cost $c^*(K)$ that is proportional to $||r||_{2/3}$ and inversely proportional to $\sqrt{K}$.

2. Expressions (6–9) highlight the importance of *spatial* distribution of requests and agree with intuition: more servers should be placed where requests concentrate.

3. The $\sqrt{K}$-dependence of the optimal cost is independent of the spatial distribution r (or normalized distribution f), but just on the distance measure. If the *square* of the Euclidean norm is used as the distance measure, then the optimal cost will be inversely proportional to K.

We illustrate with three examples with uniform, Gaussian, and power-law request densities. The requests are most widely spread with uniform density, least so with Gaussian, and in between with power-law density. For these densities, the optimal server densities are of the same type as the request densities, but fall off less sharply away from the center (see (7) or (9)).

Example 2: Uniform request density

Suppose the spatial density $f(z) = 1/|A|$ is uniform over an area A. Then the optimal server density $\lambda^*(z) = 1/|A|$ is again uniform. The optimal cost is

$$c^*(K) \;=\; \frac{2}{3}\sqrt{\frac{|A|}{\pi}} \cdot \frac{\rho}{\sqrt{K}}$$

Hence the cost is proportional to square root of the size of A. This is intuitive since the more spread-out requests are, as measured by the size of A, the more costly it is to serve them. ■

Example 3: Gaussian request density

Suppose the spatial density is Gaussian centered at the origin with independent coordinates:

$$f(z) \;=\; \frac{1}{2\pi\sigma_1\sigma_2}\, e^{-\frac{1}{2}\left(\frac{z_1^2}{\sigma_1^2}+\frac{z_2^2}{\sigma_2^2}\right)}$$

Then the optimal server density is also Gaussian centered at the origin but with $3/2$ the variance in each coordinate:

$$\lambda^*(z) \;=\; \frac{1}{2\pi\tilde{\sigma}_1\tilde{\sigma}_2}\, e^{-\frac{1}{2}\left(\frac{z_1^2}{\tilde{\sigma}_1^2}+\frac{z_2^2}{\tilde{\sigma}_2^2}\right)}$$

where $\tilde{\sigma}_i^2 = 3\sigma_i^2/2$. Hence the servers are less concentrated around the center than the requests. The optimal cost is

$$c^*(K) \;=\; \sqrt{3\sigma_1\sigma_2} \cdot \frac{\rho}{\sqrt{K}}$$

Again, the more spread out requests are, as measured by σ_1, σ_2, the more costly to serve them. ■

Example 4: Power-law request density

Suppose the spatial density has a heavy tail

$$f(z) \;=\; \alpha_1\alpha_2 z_1^{-(1+\alpha_1)} z_2^{-(1+\alpha_2)}, \quad z_i \geq 1,\ \alpha_i > 1$$

Then the optimal server density is also a power-law with a heavier tail:

$$\lambda^*(z) \;=\; \frac{1}{9}(2\alpha_1-1)(2\alpha_2-1)\, z_1^{-2(1+\alpha_1)/3} z_2^{-2(1+\alpha_2)/3}$$

and the optimal cost is

$$c^*(K) \;=\; \frac{18\alpha_1\alpha_2}{\sqrt{\pi(2\alpha_1-1)^3(2\alpha_2-1)^3}} \cdot \frac{\rho}{\sqrt{K}}$$

■

2.3 Multiple websites

Consider J websites indexed by $j = 1, 2, \ldots, J$. Suppose requests to website j has a total volume of ρ_j and a spatial density $f_j(z)$ (or equivalently, a request rate $r_j(z) = \rho_j f_j(z)$). Out of a total of K servers, k_j servers are allocated to serve website j such that

$$\sum_{j=1}^{J} k_j \;=\; K$$

and they are placed according to the optimal server density λ_j^* so that the cost associated with website j is

$$c_j(k_j) \;=\; \frac{\alpha_j}{\sqrt{k_j}}$$

where

$$\alpha_j \;:=\; \frac{2}{3\sqrt{\pi}}\,\rho_j\,||f_j||_{2/3} \;=\; \frac{2}{3\sqrt{\pi}}\,||r_j||_{2/3}$$

as explained in the last subsection. Note that servers for different websites can be co-located at the same node. We will choose server allocation k_j to minimize the network cost:

$$\begin{aligned} c^*(K) \;&=\; \min_{k_j} \sum_j c_j(k_j) \;=\; \sum_j \frac{\alpha_j}{\sqrt{k_j}} \\ \text{s. t.} \qquad & \sum_j k_j = K, \quad k_j \in \{0, 1, \ldots, K\} \end{aligned}$$

For large K, relax the constraint that k_j be integers. We hence solve the following simple convex program:

$$\begin{aligned} c^*(K) \;&=\; \min_{k_j} \sum_j \frac{\alpha_j}{\sqrt{k_j}} \\ \text{s. t.} \qquad & \sum_j k_j = K, \quad k_j \geq 0 \end{aligned}$$

The necessary and sufficient condition for k_j^* to be optimal is the Karush-Kuhn-Tucker condition:

$$\frac{\alpha_j}{2k_j^{*\,3/2}} = \beta \quad \text{for all } j$$

which, together with the feasibility condition, yields the optimal allocation and cost:

$$\begin{aligned} k_j^* &= \frac{\alpha_j^{2/3}}{\sum_l \alpha_l^{2/3}} \cdot K \\ &= \frac{\int r_j^{2/3}}{\sum_l \int r_l^{2/3}} \cdot K \qquad (10) \\ c^*(K) &= \left(\sum_j \alpha_j^{2/3}\right)^{3/2} \cdot \frac{1}{\sqrt{K}} \\ &= \frac{2}{3\sqrt{\pi}} \left(\sum_j \int r_j^{2/3}\right)^{3/2} \cdot \frac{1}{\sqrt{K}} \end{aligned}$$

Remarks:

1. Recall that ρ_j represents the popularity of website j, and f_j represents the spatial density of requests for website j. They are related through the request rate $r_j(z) = \rho_j f_j(z)$. Hence, optimal allocation depends critically on website popularities as well as spatial densities of requests. Specifically, the fraction of servers allocated to website j should be proportional to $\int r_j^{2/3}$ (or equivalently, to $\int f_j^{2/3}$). The optimal cost is proportional to $\left(\sum_j \int r_j^{2/3}\right)^{3/2}$ and inversely proportional to $\sqrt{K}$.

2. We can combine equations (9) for optimal placement and (10) for optimal allocation to express optimal number of servers in a unit area for each website j directly in terms of the total number K of servers:

$$\lambda_j^*(z) k_j^* = \frac{r_j(z)^{2/3}}{\sum_l \int r_l^{2/3}} \cdot K$$

Hence, the optimal density is proportional to $r_j(z)^{2/3}$, as a fraction of total request rate for *all* websites.

Example 5: Zipf-like popularity

It has been widely observed empirically that the popularity of websites (files) follows a Zipf-like distribution, i.e., the probability that the jth most popular website is accessed (from any node) is proportional to $j^{-\beta}$, $0 < \beta \leq 1$; see e.g. [5] and references therein. We model this by requiring the total request volume ρ_j for website j to follow a Zipf-like distribution:

$$\rho_j = Rj^{-\beta}, \quad 0 < \beta \leq 1$$

Then

$$\alpha_j = \frac{2R}{3\sqrt{\pi}} j^{-\beta} ||f_j||_{2/3}$$

Hence the number k_j^* of servers that should be allocated to website j is proportional to $(\int f_j^{2/3}) \cdot j^{-2\beta/3}$, i.e., it also follows a Zipf-like distribution. The optimal cost is

$$c^*(K) = \frac{2R}{3\sqrt{\pi}} \left(\sum_j \int (j^{-\beta} f_j)^{2/3}\right)^{3/2} \cdot \frac{1}{\sqrt{K}}$$

■

3. ALLOCATION AND PLACEMENT ALGORITHMS

The approximate model in the last section suggests a server allocation and placement algorithm where a fraction of servers that is proportional to $\int r_j^{2/3}$ is allocated to each website j, and the spatial distribution of these servers should be proportional to the $\frac{2}{3}$-power of the spatial distribution of requests. Even though the high-density model is continuous, we apply its insight to derive *discrete* algorithms.

Given:

1. I points (nodes) in $\Re^2$, indexed by $i = 1, 2, \ldots, I$.
2. J websites, indexed by $j = 1, 2, \ldots, J$;
3. $I \times J$ request rate matrix r where $r(i, j)$ represents the request rate from node i to website j;
4. the total number K of servers.

Allocation algorithm:

1. For each website j, compute $a_j := \sum_i r(i, j)^{2/3}$. Let $a := \sum_j a_j$.
2. Allocate $k_j = \max\{1, [a_j\, K/a]\}$ number of servers to website j, where $[k]$ is the nearest integer to k.[3]

Placement algorithm:

1. For each website j, place k_j servers in the following manner.
2. Partition $\Re^2$ into $N < I$ disjoint grids $A_1, A_2, \ldots A_N$.
3. For each grid A_n, compute

$$b_{jn} := \frac{k_j}{a_j} \sum_{i \in A_n} r(i, j)^{2/3}$$

Let $\lfloor b_{jn} \rfloor$ be the integer part of b_{jn} and $\check{b}_{jn} := b_{jn} - \lfloor b_{jn} \rfloor$ be the fractional part of b_{jn}.

4. Choose $\lfloor b_{jn} \rfloor$ number of nodes in grid A_n to be servers. With probability $\check{b}_{jn} \in (0, 1)$, an additional node in grid A_n is chosen to be a server.

[3] If the resulting $\sum_j k_j$ differs from K because of rounding, adjust k_j in any reasonable way to obtain $\sum_j k_j = K$.

The server allocation and placement algorithm has a time complexity of $O(IJ)$; in particular, it is linear in the number I of nodes.

We can also combine the allocation and placement algorithm into

Joint allocation and placement algorithm:

1. Compute $a := \sum_i \sum_j r(i,j)^{2/3}$.
2. Partition $\Re^2$ into $N < I$ disjoint grids $A_1, A_2, \ldots A_N$.
3. For each grid A_n, for each website j, compute
$$b_{jn} := \frac{K}{a} \sum_{i \in A_n} r(i,j)^{2/3}$$
Let $\lfloor b_{jn} \rfloor$ be the integer part of b_{jn} and $\check{b}_{jn} := b_{jn} - \lfloor b_{jn} \rfloor$ be the fractional part of b_{jn}.
4. Choose $\lfloor b_{jn} \rfloor$ number of nodes in grid A_n to be servers for website j. With probability $\check{b}_{jn} \in (0,1)$, an additional node in grid A_n is chosen to be a server for website j.

4. PERFORMANCE EVALUATION

In this section we present preliminary simulations to compare the performance of our algorithm with the approximation algorithm of [12]. Since the approximation algorithm does not dictate how servers should be allocated when there are multiple websites, we study only the case of single website.

As noted above, even though the insight of high-rate vector quantization is obtained in a continuous model, the server allocation and placement algorithm works on discrete problems with arbitrary (discrete) request rates r. However, to compare with the analytical (approximate) expression for costs, we choose in our simulations request distributions whose continuum approximations are uniform, Gaussian, and power-law, the three spatial request densities in Examples 2-4, as follows.

For each simulation run, we first generate I points in the unit square $[0,1]^2$, uniformly distributed over the unit square. We fix a total request volume ρ. For each point i, we assign its request rate to be
$$r(i) = \frac{\rho f(i)}{I}$$
where $f(i)$ is the spatial density over the square $[0,1]^2$ evaluated at the *coordinate* $z_i := (x_i, y_i)$ of point i. Then $f(i)$ is a continuum (and normalized) approximation of $r(i)$, in the sense that $r(i)$ represents the demand in a small region of size $1/I$ around z_i that follows the spatial density f.

We take the number K of servers to be 20% of I. We execute our server placement algorithm on the given set of I nodes, $J = 1$ website, and request vector r to obtain the coordinates of these K servers. We vary I from 100 to 20,000, and measure the network cost
$$c(I) = \sum_{k=1}^{K} \sum_{i \in V_k} r(i) ||z_i - s_k||$$
where $z_i = (x_i, y_i) \in [0,1]^2$ is the coordinate of point i and $s_k \in [0,1]^2$ is the coordinate of the nearest server k. It is simpler to look at the normalized cost
$$\hat{c}(I) = \frac{\sqrt{K}}{\rho} \cdot c(I)$$
which is predicted by the theory to be a constant independent of I, K or ρ, and depends only on the spatial density f (see the expressions for optimal costs in Examples 2, 3, and 4).

We note that the for each case of I, the number of servers assigned by our algorithm may be different from K $(= 0.2I)$, because of two reasons. First, the algorithm computes the number of servers that should be located in a grid A_n, and each server must be located at a node in that grid. For a grid A_n with a large aggregate request rate $\sum_{i \in A_n} r(i)$, the number of servers assigned to A_n may exceed the number of points in A_n. This depends on the size of the grid. The excess is ignored when all points in grid A_n have been assigned as servers. Second, the fractional part of the optimal number of servers is used to add a server probabilistically. Hence, for each I, we conduct 20 simulation runs, and for each run, we note the coordinates of the I points and the actual number of assigned servers.

We then apply the k-median algorithm of [12] to each of the 20 simulation runs, using the same coordinates and the actual number of servers produced by our algorithm for each run.

The average normalized costs $\hat{c}(I)$, averaged over the 20 runs, are shown in Figures 1–3 as a function of I. For each spatial distribution, we plot the normalized costs of our algorithm (for I up to 20,000), and that of the k-median algorithm (for I up to 1,000). Also shown in the figures are the theoretical values for the normalized costs according to the high density model (straight lines).

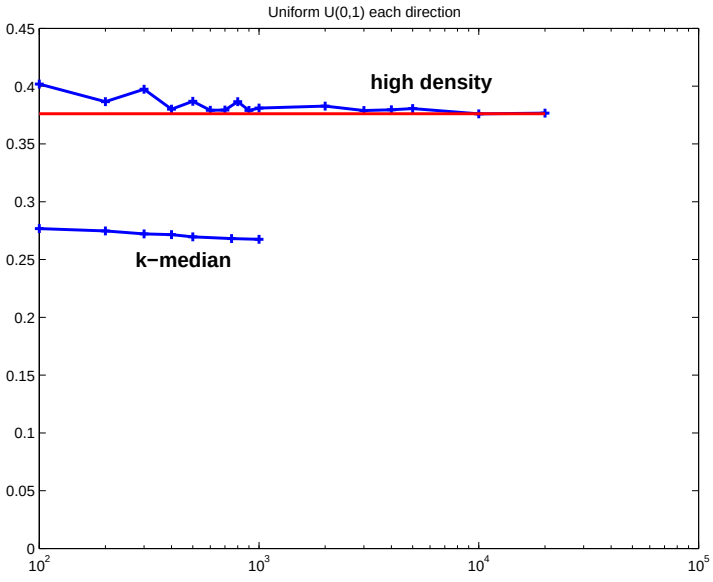

Figure 1: Normalized cost $\hat{c(I)}$ as a function of the number I of nodes for uniform request distribution.

We make a few remarks. First, the empirical costs of our algorithm are close to the analytical value, validating our high density model. Second, the cost under our algorithm is never more than twice the cost under the k-median algorithm. Third, our algorithm has time

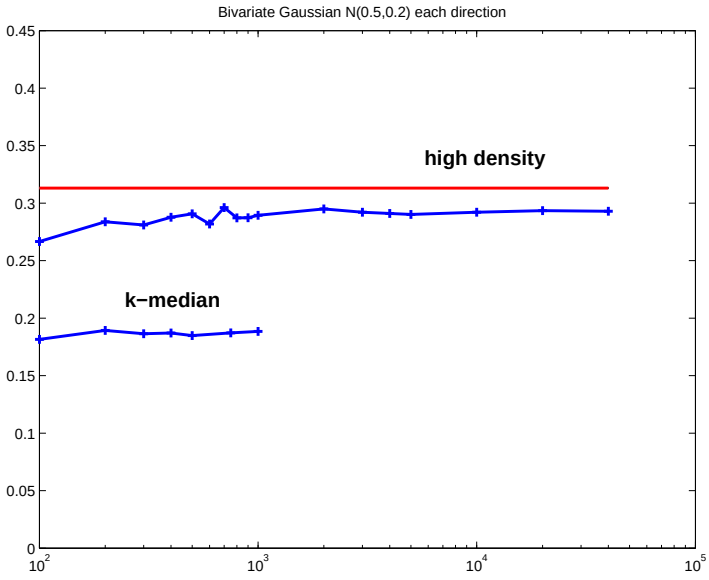

Figure 2: Normalized cost $c(\hat{I})$ as a function of the number I of nodes for Gaussian request distribution.

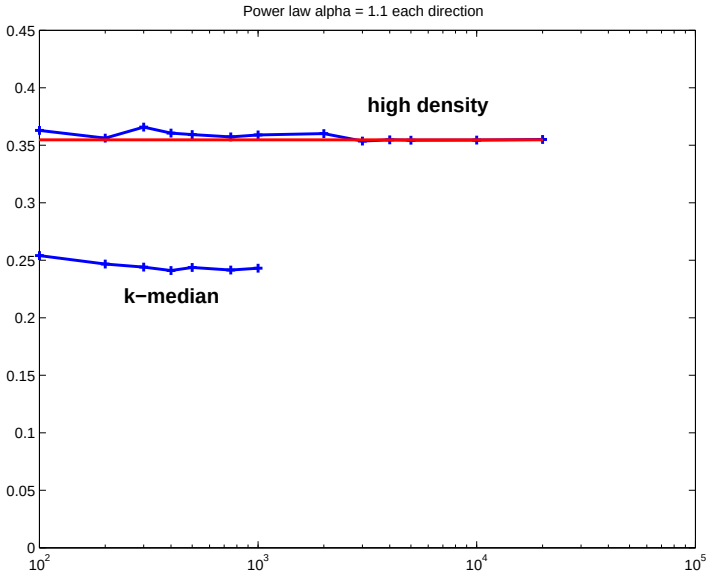

Figure 3: Normalized cost $c(\hat{I})$ as a function of the number I of nodes for power-law request distribution.

I	High density	k-median
100	0.01 sec	1.3 sec
200	0.01	5.6
500	0.02	53
750	0.02	144
1000	0.03	294
5000	0.17	—
10000	0.35	—
20000	0.69	—

Table 1: Average running time

complexity linear in I and hence can handle much larger problem sizes. This tradeoff may be appropriate for large-scale dynamically self-organizing CDN.

Table 1 shows the running time, averaged over the 20 runs for each I. The simulations were run on a 1.5GHz Pentium 4 processor with 256Mb RAM, linux kernel 2.4.2, gcc 2.96-79 and Matlab R12.1.

5. CONCLUSION

By viewing server placement as a vector quantization problem in the case when both clients and servers are dense, we have derived a simple server allocation and placement algorithm. Preliminary simulations suggest that it has a good tradeoff between performance and complexity. Our results highlight the importance of both the *spatial* distributions of client requests for each website and the total request volume for each website. The total volume represents the website's popularity and has been found to follow a Zipf-like distribution [5]. It determines the server allocation: more popular contents should be allocated more servers. The spatial distribution of requests determines where these servers should be placed around the network. However spatial distribution of requests is not well exploited in current systems, both because of the difficulty in measuring it empirically [16, 13, 25], and because of the lack of a theoretical understanding of its role.

Our current joint sever allocation and placement algorithm, though much simpler, still requires nonlocal information to implement. The idea is to have each local region decide how many servers to employ and what contents to store based only on the fraction of local client requests for different contents, as a fraction of total volume on the network, a piece of nonlocal information. The scheme naturally adapts to changes in client request pattern to reduce propagation delay and balance network load. It'd be interesting to develop a distributed version of this algorithm that is suitable for real-time implementation on a large scale.

Acknowledgment. We are grateful to Michelle Effros, Vangelis Markakis and Vijay V. Vazirani for helpful discussions.

6. REFERENCES

[1] Vincenzo Auletta, Domenica Parente, and Giuseppe Persiano. Dynamic and static algorithms for optimal placement of resources in a tree. *Theoretical Computer Science*, 165:441–461, 1996.

[2] Y. Bartal. Probabilistic approximation of metric spaces and

its algorithmic applications. In *Proc. 37th IEEE Symp. on Foundation of Computer Science*, pages 184–193, 1996.

[3] S. Bhattacharjee, K. Calvert, and E. W. Zegura. Self-organizing wide-area network caches. In *Proceedings of IEEE Infocom*, March/April 1998.

[4] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In *Proc. of the Second International Conf. on the World Wide Web*, pages 763–771, 1994.

[5] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE Infocom*, March 1999.

[6] Anawat Chankhunthod, Michael Schwartz, Peter Danzig, Kurt Worrell, and Chuck Neerdaels. A hierarchical internet object cache. In *Proceedings of the 1996 Usenix Technical Conference*, pages 153–163, 1996.

[7] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median. In *Proc. 30th ACM Symp on Theory of Computing*, pages 114–123, 1998.

[8] M. Charikar, S. Guha, E. Tardos, and D. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, May 1999.

[9] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. on Networking*, 8(3):281–293, June 2000.

[11] Robert M. Gray. *Source coding theory*. Kluwer Academic Publishers, 1990.

[12] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *to appear in J. ACM*, 2001.

[13] J. Kangasharju, K. W. Ross, and J.W. Roberts. Locating copies of objects using the domain name system. In *Proceedings of the 4th International Caching Workshop*, March 1999.

[14] D. Karger, T. Leighton, D. Lewin, and A. Sherman. Web caching with consistent hasing. In *Proc. of the 8th Int World Wide Web Conf.*, May 1999.

[15] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. II: the p-medians. *SIAM J. Appl. Math.*, 37(3):539–560, December 1979.

[16] Balachander Krishnamurthy and Jia Wang. On network-aware clustering of web clients. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[17] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Trans. on Networking*, 8(5):568–582, October 2000.

[18] Bo Li, Mordecai Golin, Giuseppe Italiano, Xin Deng, and Kazem Sohraby. On the optimal placement of web proxies in the internet. In *Proceedings of IEEE Infocom*, March 1999.

[19] B. S. Michel, K. Nikoloudakis, P. Reiher, and L. Zhang. URL forwarding and compression in adaptive web caching. In *Proc. of IEEE Infocom*, volume 2, pages 670–678, March 2000.

[20] Scott Michel, Khoi Nguyen, Adam Rosenstein, Lixia Zhang, Sally Floyd, and Van Jacobson. Adaptive Web caching: towards a new global caching architecture. *Computer Networks and ISDN Systems*, 30(22–23):2169–2177, 1998.

[21] C. Papadimitriou. Worst-case and probabilistic analysis of a geometric location problem. *SIAM J. Comput.*, 10:542–557, 1981.

[22] L Qiu, V. Padmanabhan, and G. Voelker. On the placement of web server replicas. In *Proceedings of IEEE Infocom*, April 2001.

[23] M. Rabinovich and A. Aggarwal. RaDaR: A scalable architecture for a global web hosting service. In *Proceedings of the 8th Int. World Wide Web Conf.*, May 1999.

[24] K. W. Ross. Hash-routing for collections of shared web caches. *IEEE Network Magazine*, pages 37–45, November/December 1997.

[25] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proceedings of IEEE Infocom*, April 2001.

[26] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Oper. Res. Lett.*, 19:59–64, 1996.

[27] J. Touch. The LSAM proxy cache - a multicast distributed virtual caceh. In *Proc. of the 3rd Int. WWW Caching Workshop*, June 1998.

[28] Duane Wessels and K. Claffy. Icp and the squid web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, April 1998.