Optimal Client-Server Assignment for Internet Distributed Systems

Hiroshi Nishida
School of EECS, Oregon State University
Corvallis, OR 97331-5501, USA
Email: nishidah@onid.orst.edu

Thinh Nguyen
School of EECS, Oregon State University
Corvallis, OR 97331-5501, USA
Email: thinhq@eecs.oregonstate.edu

***Abstract*—We investigate an underlying mathematical model and algorithm for optimizing the performance of a class of distributed systems over the Internet. Such a system consists of a large number of clients who communicate with each other indirectly via a number of intermediate servers. Optimizing the overall performance of such a system then can be formulated as a client server assignment problem whose aim is to assign the clients to the servers in such a way to satisfy some pre-specified requirements on the communication cost and load balancing. Under the proposed mathematical model, we show that 1) the total communication load and load balancing are two opposing metrics, and consequently, their trade-off is inherent to this class of distributed systems; 2) in general, finding the optimal client-server assignment for some pre-specified requirements on the total load and load balancing is NP-hard, and therefore; 3) we propose a heuristic via relaxed convex optimization for finding the approximate solution to the client-server assignment problem. Our simulation results indicate that the proposed algorithm produces superior performance than other heuristics, including the popular Normalized Cuts algorithm.**

I. Introduction

An Internet distributed system consists of a number of nodes (e.g., computers) that are linked together in ways that allow them to share resources and computation. An ideal distributed system is completely decentralized, and that every node is given equal responsibility and no node is more computational or resource powerful than any other. However, for many real-world applications, such a system often has a low performance due to a significant cost of coordinating the nodes in a completely distributed manner. In practice, a typical distributed system consists of a mix of servers and clients. The servers are more computational and resource powerful than the clients. A classical example of such systems is Email. When a client A sends an email to another client B, A does not send the email directly to B. Instead, A sends its message to its email server which has been previously assigned to handle all the emails to and from A. This server relays A's email to another server which has been previously assigned to handle emails for B. B then reads A's email by downloading it from its local server. Importantly, the email servers communicate with each other on behalf of their clients. The main advantage of this architecture is *specialization*, in the sense that the powerful dedicated email servers release their clients from the responsibility associated with many tasks including processing and storing emails, and thus making email applications more scalable.

Email systems assign clients based primarily on the organizations that the clients belong to. Two employees working for the same company will have their email accounts assigned to the same email server. Thus, the client server assignment is trivial. A more interesting scenario is the Instant Messaging System (IMS). IMS allows real-time text-based communication between two or more participants over the Internet. Each IMS client is associated with an IMS server which handles all the instant messages for its clients. Similar to email servers, IMS servers relay instant messages to each other on behalf on their clients. In an IMS that uses the XMPP (Jabber) [1] protocol, clients can be assigned to servers independent of their organizations. Furthermore, the client-server assignment can be made dynamic as deemed suitable, and thus making this problem much more interesting.

In XMPP, a username is set as *user@domain* (e.g., *nishida@jabber.org*) just like an email account, where *domain* usually stands for a server name in which *user* is registered. When a user (*aaa@domain*) sends a message to another user in the same domain (*bbb@domain*), the message is delivered only through the *domain* server (i.e., *aaa* → *domain* server → *bbb*). The clients do not directly exchange their messages each other. When a user (*aaa@domain1*) sends a message to another user in a different domain (*bbb@domain2*), the message is sent as: *aaa* → *domain1* server → *domain2* server → *bbb*. This design is indeed simple and scalable. If the number of users increases, another server can be added to accommodate the new users. That said, we want to consider how to optimally assign clients to servers, beginning with the following observations:

- If two users who exchange many messages with each other are assigned to two different servers, then the amount of communication between their servers increases, and thus the overall load due to communication increases. On the other hand, if these two users are assigned to the same server, all their communication are local, thus it is more efficient. So it makes sense to assign all the clients to as few servers as possible.
- However, if we have multiple servers and try to use as few number of servers as possible, it may happen that some servers are heavily loaded while others are not. Since a heavily loaded server typically exhibits a low performance, we would like to avoid this situation. There-

fore, we want a client-server assignment that produces an overall fair load balance among the servers.

Given the observations above, we must strike a balance between reducing the overall communication load, equivalently the communication cost among the servers, and increasing the load fairness among the servers, i.e., the load balance. *The primary contribution of this paper is a heuristic algorithm via relaxed convex optimization that takes a given communication pattern among the clients as an input, and produces an approximately optimal client-server assignment for a pre-specified trade-off between load balance and communication cost.* Next, we describe a number of emerging applications that have the potential to benefit from the client-server assignment problem.

A. Emerging Applications

The client-server assignment problem is also relevant to a host of emerging applications ranging from social network applications such as Facebook and Twitter to online distributed auction systems such as eBay. Facebook is a system that allows circles of friends to exchange messages and pictures among themselves. Since friends are likely to communicate with each other than non-friends, assigning friends to the same server will reduce the inter-server communication. At the same time, it is preferable to balance the communication load. This is exactly the client-server assignment problem encountered in the IMS.

Online distributed auction systems is another candidate for applying the client-server assignment. If a user logged in a server which has content that is not of interest to the user, it will generate the communication overhead between multiple servers every time the user searches for an item. Therefore, letting a user log in the server that has contents that the user is interested in will raise the efficiency. In this case, the contents (or the categories of the contents) also need to be classified as clients.

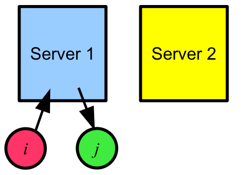
The client-server assignment also has the potential to be applicable to distributed database systems, such as MapReduce [2]. Assigning the search keywords which are often queried together to the same servers will reduce the inter-server communication. In this case, the search keywords correspond to the clients in the above instant messenger systems.

B. Model for Communication

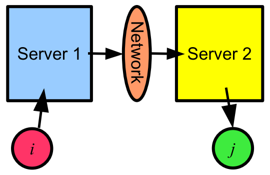
We model the load incurred on a server for sending and receiving messages. We call the load incurred by communication the communication load. This model will be important for the precise formulation of the client-server assignment problem in Section III.

We begin with an example shown in Fig. 1. The IMS of interest operates according to the following rules:

a) If client i and j are logged in the same server and exchange messages each other, then their messages are processed only through the server (Fig. 1(a)).
b) If client i and j are logged in different servers (i is in server 1, j is in server 2) and suppose i sends a message to j, then server 1 forwards the message to server 2 and server 2 sends it to j (Fig. 1(b)). This causes an extra communication overhead compared to case a), because it generates an additional message passing between servers 1 and 2.



(a) Client i and j are logged in server 1. The messages between them are passed only via server 1.



(b) Client i is logged in server 1 and client j is logged in server 2. The messages are passed through server 1 and 2, which causes extra communication.

Fig. 1. Example of client assignment to servers

	Server 1	Server 2
i and j are in server 1	Receive from i, send to j	
i is in server 1 j is in server 2	Receive from i, send to 2	Receive from 1, send to j

TABLE I
MESSAGE PROCESSING SUMMARY IN FIG. 1

We assume that all messages are encrypted and the server load by communication is proportional to the amount of messages sent from and received by the server. Specifically,

- If both clients i and j are logged in server 1, server 1 receives the messages from i and sends them to j.
- If client i is logged in server 1 and j is in server 2, server 1 receives the messages from i and sends them to server 2, then server 2 receives them and sends to j.

As a result, the message processing can be summarized in Table I. As seen, the amount of processing by server 1, i.e., the communication load, does not change by assigning client j to server 2. Only the difference is the receiver to whom server 1 sends the messages; if both client i and j are logged in server 1, then the receiver is client j, and if j is assigned to server 2, then the receiver is server 2. Furthermore, by assigning j to server 2, the new load is generated on server 2, whose amount is the same as the one on server 1. If clients i and j seldom contact each other, assigning them to different servers will minimally increase the overall communication load. On the other hand, if they contact frequently, separating them increases the communication load. Thus, assigning them to the same server is more efficient. In most distributed systems however, load balance is an important consideration. If all clients are assigned to a single server, the amount of inter-server communication is zero. Although this is optimal in terms of the communication overhead, it implies that one server does all the work while others are idle, resulting in an extreme load unbalance, leading to degraded performance.

II. Related Work

Clustering Algorithms. To a certain extent, the client-

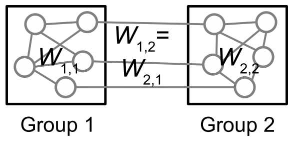


Fig. 2. Example of Normalized Cuts

server assignment problem can be viewed as an instance of the clustering problem. Specifically, the clients and their communication patterns can be represented as a graph whose vertices denote the clients, an edge between two vertices denote a communication between two corresponding clients. The weight of an edge between two vertices represents how frequent the two clients communicate with each other. The goal of many clustering algorithms is to cluster the clients into a fixed number of groups so that a certain objective, e.g., the ratio of inter-communication among groups to intra-communication within a group, is minimized. Therefore, we briefly discuss a few approaches to the clustering problem. The most related clustering algorithm to our problem is the *Normalized Cuts* (NC) [3] that partitions an undirected graph into two disjoint partitions so that

$$F_{ncut} = \frac{W_{1,2}}{W_{1,1}+W_{1,2}} + \frac{W_{2,1}}{W_{2,2}+W_{2,1}} \quad (1)$$

is minimized, where $W_{i,j}$ is the sum of the weights of all edges that connect the vertices in group i and j (see Fig. 2). The less the amount of inter-group communication and the more balanced the volumes of the groups (volume = the sum of the weights of all edges in the group), the less F_{ncut} we have. In this sense, F_{ncut} is very similar to our objective (see Section III-C). NC utilizes the eigenvalues of the adjacency matrix and solves (1) efficiently. The NC is especially suitable for segmenting an image, and is also widely used in bioinformatics and machine learning communities.

Different from methods such as [4] that minimizes the largest inter-group flow, the NC considers balancing the volumes of the groups. However, the NC tends to isolate vertices which do not have strong connection to others and causes unbalance in the volumes of the groups, especially in the power-law graphs [5] [6] [7]. In [6], the authors examine some balanced-clustering algorithms for power-law graphs. Interestingly, [6] simulates with a graph based on the buddy lists of Yahoo IM, which is also an instant messenger system, and concludes that the combination of solving a semidefinite program and multiple tries of a randomized flow-based rounding methods yields effective results. Though this seems relevant to our research, there is a significant difference. [6] focuses on balancing each group size which is *the number of vertices* in a group, while our research requires to balance *the sum of weights of inner- and inter-group edges* in each group which corresponds to the communication load in our case (see Section III-B). Therefore, [6]'s research cannot be directly adopted to our problem. To the best of knowledge, there is no clustering algorithm which achieves our goal: minimizing the communication between servers and balancing the communication load.

Load Balancing Distributed Virtual Environment Systems. In [8] [9] [10], efficient client-server assignment for distributed virtual environment (DVE) systems is studied. The DVE systems allow multiple users working on different client computers to interact in a shared virtual world. The DVE systems are generally operated by multiple servers, and therefore exhibit the same set of issues as ours: balancing the workload and reducing the communication between the servers. However, unlike our problem, their overall workload is assumed to be constant regardless of the client-server assignment.

III. Optimal Client-Server Assignment

As discussed in Section I, the total communication load and load balance are two opposing metrics. Thus, different applications will allow for different trade-offs between these two quantities. Our goal in this section is to derive the expressions for the total communication load and the load balance for a given communication pattern among the clients. Based on these, we will formulate a mathematical optimization problem for this trade-off. We begin with the notations.

A. Notation

The following is the notation used in this paper for vector v and matrix A:

$\lvert v\rvert_1$:	Norm-1 of the vector v, i.e., the sum of all elements in v.
$\lVert A\rVert_1$:	Elementwise norm of matrix A, i.e., the sum of all elements in A.

Also, we define the followings parameters:

M:	The number of servers in the system.
N:	The number of clients in the system, $N > M$.
S:	A $[0,1]^{N\times N}$ matrix whose element $S_{i,j}$ represents the rate of messages sent from client i to j in the system. S represents the communication patterns among the clients. Note $\lVert S\rVert_1 = 1$ and in many systems, we will have $S_{i,i} = 0\ \forall i$ because messages sent to itself will be processed by a client software, not through a server. Alternatively, if two clients i and j are selected uniformly at random, $S_{i,j}$ can be viewed as the probability that client i sends a message to client j. As a result, S can be viewed as the distribution on the ordered pair of clients.
X:	An unknown matrix, $X \in \{0,1\}^{N\times M}$ where $X_{i,s} = 1$ if client i is assigned to server s and $X_{i,s} = 0$ otherwise. Since a client is assigned to only one server, $\sum_{s=1}^{M} X_{i,s} = 1\ \forall i$. We can view each row of X as a point mass distribution.

Next we will derive the expressions for the communication load, i.e., the number of messages processed by a server in terms of S, the client communication pattern and X, the client-server assignment.

B. Communication Load

Let $P_{s,t}$ represent the rate of messages sent from server s to t, then we have:

$$P_{s,t} = \sum_{i=1}^{N}\sum_{j=1}^{N} S_{i,j} X_{i,s} X_{j,t}, \tag{2}$$

that is:

$$P = X^T S X, \tag{3}$$

where $P \in [0,1]^{M\times M}$ and $\|P\|_1 = 1$. Similar to S, if two servers s and t are selected uniformly at random, $P_{s,t}$ can be viewed as the probability that server s sends a message to server t, and consequently P can be viewed as the distribution on the ordered pair of servers.

As described in Section I, when messages are passed between two clients only through a single server, i.e., the two clients are assigned to the same server, the amount of processing on the server is $1\times$(receive+send). However, when the two clients are assigned to different servers, the amount of processing is $1\times$ (receive+send) for each server and $2\times$ (receive+send) in total (see Table I). Thus, to calculate the communication load, two different types of message passing should be considered:

1) Message passing through a single server, i.e., intra-server communication.
2) Message passing through two servers, inter-server communication.

The communication load for 1) is proportional to $P_{s,s}$. The communication load for 2) is proportional to $P_{s,t} + P_{t,s}$ (for each server of s and t) because both sending and receiving causes message processing. As a result, let $L \in [0,1]^{M\times M}$ represent the load generated by the message exchanges, then

$$L = P + P^T - P^D, \tag{4}$$

where P^T is the transpose of P and P^D is the diagonal matrix of P (i.e., $P^D_{s,s} = P_{s,s}\ \forall s$ and $P^D_{s,t} = 0\ \forall s \neq t$). Note L is symmetric and $1 \leq \|L\|_1 \leq 2$.

Since $P = X^TSX$, $P^T = (X^TSX)^T = X^TS^TX$, $P^D = (P+P^T)^D/2 = (X^TSX + X^TS^TX)^D/2$, we have:

$$L = X^TSX + X^TS^TX - \frac{1}{2}(X^TSX + X^TS^TX)^D. \tag{5}$$

Let $A = S + S^T$, we have:

$$Q = X^T A X \tag{6}$$

$$L = Q - \frac{1}{2}Q^D \tag{7}$$

Note A ($\in [0,1]^{N\times N}$) is symmetric and $A_{i,j} = A_{j,i}$ can be interpreted as the rate of messages 'exchanged' (= sent + received) between clients i and j. Also, $\|A\|_1 = \|Q\|_1 = 2$.

As a result, let $l \in [0,1]^M$ be a vector denoting the communication load for M servers, then

$$l = L\mathbf{1}, \tag{8}$$

where $\mathbf{1}$ denotes a column vector whose all elements are 1.

C. Metrics

In this subsection, we will define the total communication load and load balance, the two important metrics to be used in our optimization problem.

Total Communication Load. Total communication load is the total load on all servers which can be defined as:

$$\begin{aligned}\|L\|_1 &= \|Q - \tfrac{1}{2}Q^D\|_1 && (9)\\ &= \|\tfrac{1}{2}Q + \tfrac{1}{2}(Q - Q^D)\|_1 && (10)\\ &= 1 + \|\tfrac{1}{2}(Q - Q^D)\|_1 && (11)\\ &= 1 + \textstyle\sum_{s=1}^{M}\sum_{t=1}^{s-1} L_{s,t}. && (12)\end{aligned}$$

Let

$$F_c = \sum_{s=1}^{M}\sum_{t=1}^{s-1} L_{s,t}, \tag{13}$$

then F_c is the the sum of non-diagonal entries of L divided by 2, and thus represents the amount of inter-server communication. The total communication load equals to the intra-server communication load plus the inter-server communication load. From the above equation, the intra-server communication load is a constant, i.e., 1, only the inter-communication load F_c is optimized. Note that $0 \leq F_c \leq 1$. The smaller F_c results less total communication load.

Load Balance. Intuitively, load balance should be a metric that represents the degree of load variations among different servers. Some popular metrics are variance, entropy, and *Gini coefficient*. The Gini coefficient is used often in economics to measure the inequality of income distribution in a society. In this paper, we consider the Gini coefficient as a load balance metric as it empirically captures the requirements of load balance on the servers better than other metrics. Specifically, for large M, the Gini coefficient is more sensitive to a slight change in the load balance than the entropy and variance.

Mathematically, in the context of the total server load, the Gini coefficient is defined as:

$$\begin{aligned}F_l &= \tfrac{M}{M-1}\left(\tfrac{2\sum_{s=1}^{M} s\, l_s}{M\sum_{s=1}^{M} l_s} - \tfrac{M+1}{M}\right) && (14)\\ &= \tfrac{1}{M-1}\left(\tfrac{2\sum_{s=1}^{M} s\, l_s}{\sum_{s=1}^{M} l_s} - M - 1\right), && (15)\end{aligned}$$

where $l_1 \leq l_2 \leq \cdots \leq l_M$. As mentioned, F_l is scaled to $0 \leq F_l \leq 1$, and the smaller F_l is, the better load balance we have.

To see why the Gini coefficient is more sensitive to a slight change in the load balance than the entropy and variance, we consider the following example. If $M = 10$ and $\frac{l_s}{\|l\|_1} = \frac{1}{10}\ \forall s$ (i.e., the uniform distribution), then we have the entropy $-\sum_{s=1}^{M} \frac{l_s}{\|l\|_1} \log_M \frac{l_s}{\|l\|_1} = 1$, the variance $(\frac{M}{M-1})^2 \sum_{s=1}^{M} (\frac{l_s}{\|l\|_1} - \frac{1}{M})^2 = 0$ and the Gini coefficient (15) = 0, where the metrics are all scaled to $[0,1]$. However, if $\frac{l_1}{\|l\|_1} = 0$, $\frac{l_{10}}{\|l\|_1} = \frac{1}{5}$, $\frac{l_s}{\|l\|_1} = \frac{1}{10}$ for $2 \leq s \leq 9$, then we have the entropy = 0.94, the variance = 0.025, the Gini coefficient = 0.2, and the corresponding differences are 0.06, 0.025 and 0.2 respectively.

In a real distributed system, $\frac{l_1}{\|l\|_1} = 0$ i.e., no load on server 1 is supposed to be a serious issue, but it is not sufficiently reflected when using the variance and entropy as metrics.

D. Problem Formulation and Hardness Result

After deriving the expressions for communication load F_c and load balance F_l in terms of client communication patterns and a client-server assignment, we are now ready to formulate our optimization. Let

$$F = \alpha F_c + (1-\alpha) F_l, \tag{16}$$

where $0 \le \alpha \le 1$ is an arbitrary coefficient. We want to minimize F. Note that $0 \le F \le 1$, and the smaller F is, the more optimal the system is. The value of α is set to select a certain trade-off between load balance and total communication load; if one places more importance on reducing the total communication load, α should be large. To simplify our discussion, we use $\alpha = 0.5$ in the rest of the paper, namely:

$$F = 0.5F_c + 0.5F_l. \tag{17}$$

Formally, our optimization problem is cast as:

$$\begin{array}{rl} \text{Minimize} & F \\ \text{Subject to} & X \in \{0,1\}^{N\times M},\ \sum_{j=1}^{M} X_{i,j} = 1\ \forall i. \end{array} \tag{18}$$

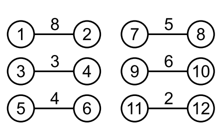
Note that our optimization problem is one of many optimization problems that we can formulate after having the mathematical expressions for load balance and total communication load.

Proposition 3.1: Our optimization problem in (18) is NP-hard, i.e., there is no polynomial time algorithm to find the optimal solution unless $P = NP$.

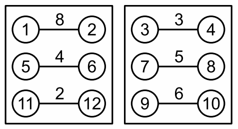
Proof: (Proof Sketch) The main idea is to show that the well-known *partition problem* is an instance of our problem. The partition problem is to decide whether a given set of integers can be partitioned into two sets with identical sums. For a given set of integers, we can always construct a corresponding special graph that represents the communication pattern of the clients such that an optimal partition of integers into two sets will result in the optimal client-server assignment.

Specifically, if there are K integers in a set, we will construct $N = 2K$ clients, each client is to communicate with exactly one other client. Therefore, there is a total of K edges connecting between K pairs of clients. We can assign the weight of an edge between a pair of clients as exactly one of the integers in the given set of the integers. Fig. 3(a) shows the construction of such a graph for the given set of integers $\{8, 3, 4, 5, 6, 2\}$. We let the number of servers $M = 2$. Clearly, $F_c = 0$.

If we find an optimal partition, i.e., one that minimizes F_l, or equivalently, the one that makes the sum of the edges in the two partitions equal, then clearly this is also a solution to the partition problem. In this example, our problem is equivalent to splitting $\{8, 3, 4, 5, 6, 2\}$ into two groups with equal sums as shown in Fig. 3(b). Since the partition problem is known to be NP-complete, our problem is NP-hard. ■



(a) Example of graph



(b) Optimal clustering

Fig. 3. Special case of our problem is equivalent to partition problem

IV. Approximate Methods via Relaxed Convex Optimization

Since our problem is NP-hard, in this section we present an approximation method via relaxed convex optimization. The main idea of our approach is to solve the special case with the number of servers $M = 2$ via relaxed convex optimization. Specifically, we will approximate both the objective and the solution domain with convex functions and a convex set, respectively. Next, we show how to apply this result to the general case for $M > 2$. The main idea is to split the servers into two groups sequentially. For each group of servers, we then recursively solve the problem for $M = 2$. Empirical results show that this method approximates the optimal solution very well.

A. Two-Server Solution

Suppose there are only two servers and x is a $\{0,1\}^N$ vector whose element x_i indicates that client i is assigned to server $x_i + 1$ (so, if $x_i = 0$, i is assigned to server 1, if $x_i = 1$, i is assigned to server 2). Then, the amounts of inter- and inner-server messages are:

$$L^B = \begin{pmatrix} \frac{1}{2}(\mathbf{1}-x)^T A(\mathbf{1}-x) & (\mathbf{1}-x)^T Ax \\ x^T A(\mathbf{1}-x) & \frac{1}{2}x^T Ax \end{pmatrix}, \tag{19}$$

where $\mathbf{1}$ is a vector with N ones, $L^B_{1,1} = \frac{1}{2}(\mathbf{1}-x)^T A(\mathbf{1}-x)$ is the amount of messages exchanged only through server 1, $L^B_{1,2} = (\mathbf{1}-x)^T Ax = L^B_{2,1} = x^T A(\mathbf{1}-x)$ is the amount of messages exchanged between server 1 and 2, $L^B_{2,1} = \frac{1}{2}x^T Ax$ is the amount of messages exchanged only through server 2. Suppose D is a diagonal matrix such that $D_{i,i} = \sum_{j=1}^{N} A_{i,j}\ \forall i$, then we have $(\mathbf{1}-x)^T Ax = x^T A(\mathbf{1}-x) = x^T(D-A)x$, that is, the amount of inter-server communication can be expressed as:

$$F^B_c = x^T(D-A)x, \tag{20}$$

which is equivalent to F_c (13) for $M = 2$ and $0 \le F^B_c \le 1$. Note $(D-A)$ is a Laplacian matrix and therefore is symmetric positive semidefinite. Hence, F^B_c is a convex function, and the smaller F^B_c is, the less inter-server communication we have.

Also,

$$\tfrac{1}{2}(\mathbf{1}-x)^T A(\mathbf{1}-x) = \tfrac{1}{2}\{d^T(\mathbf{1}-x) - F^B_c\} \tag{21}$$

$$\tfrac{1}{2}x^T Ax = \tfrac{1}{2}(d^T x - F^B_c), \tag{22}$$

where $d = \mathbf{1}^T A = A^T\mathbf{1}$ is a vector composed of D's diagonal elements and $|d|_1 = \|A\|_1 = 2$. Consequently, we have:

$$L^B = \begin{pmatrix} \frac{1}{2}\{d^T(\mathbf{1}-x) - F^B_c\} & F^B_c \\ F^B_c & \frac{1}{2}(d^T x - F^B_c) \end{pmatrix}, \tag{23}$$

and the communication loads are:

$$l_1 = L^B_{1,1} + L^B_{1,2} = \tfrac{1}{2}\{d^T(\mathbf{1}-x) + F^B_c\}, \quad (24)$$
$$l_2 = L^B_{2,1} + L^B_{2,2} = \tfrac{1}{2}(d^T x + F^B_c). \quad (25)$$

Based on (24) (25), we propose two convex functions that approximate our original objective function, i.e., the Gini coefficient.

The first convex function is based on the difference between l_1 and l_2. Since $l_1 - l_2 = \frac{1}{2}(|d|_1 - 2d^T x)$ and $|d|_1 = 2$, $(l_1 - l_2)^2$ i.e.,

$$F^B_{ld} = (1 - d^T x)^2 \quad (26)$$

can be utilized as a new load balance metric. Note F^B_{ld} is convex and $0 \le F^B_{ld} \le 1$. Since the Gini coefficient herein is $|l_1 - l_2|/(l_1 + l_2) = |l_1 - l_2|/(1 + F^B_c)$ and we also have to minimize F^B_c, minimizing (26) approximately minimizes the Gini coefficient.

The second convex function is based on the entropy of l_1 and l_2. In (24) (25), F^B_c is common for both l_1 and l_2. Therefore, in order to balance l_1 and l_2, balancing $d^T(\mathbf{1}-x)$ and $d^T x$ is enough. Consequently, we can use the following minus entropy function as another load balance metric:

$$F^B_{le} = \tfrac{d^T(\mathbf{1}-x)}{2} \log_2 \tfrac{d^T(\mathbf{1}-x)}{2} + \tfrac{d^T x}{2} \log_2 \tfrac{d^T x}{2}. \quad (27)$$

Since $\frac{d^T(\mathbf{1}-x)}{2} + \frac{d^T x}{2} = \frac{|d|_1}{2} = 1$, F^B_{le} is also convex and $0 \le F^B_{le} \le 1$. The smaller F^B_{le} is, the better load balance. In an ideal case, both the negative of entropy and the Gini coefficient are minimized when the server load distribution is uniform. Thus, we approximate Gini coefficient with the negative entropy function which is convex.

As a result, (20) + (26) and (20) + (27), i.e.,

$$F^B_d = \beta F^B_c + (1-\beta) F^B_{ld} \quad (28)$$
$$F^B_e = \beta F^B_c + (1-\beta) F^B_{le} \quad (29)$$

become our new metrics, and optimal solutions are obtainable by minimizing them, because both are convex functions. Note β ($0 \le \beta \le 1$) is an arbitrary coefficient to balance F^B_c and F^B_{ld} (or F^B_{le}), and $\beta = \alpha$ is thought to be reasonable.

Next, we relax the constraint of x_i being binary, to allow $x_i \in [0, 1]$. However, (28) (29) with a weak constraint such as $0 \le x \le 1$ output $x_1 = x_2 = \cdots = x_N = 0.5$, which is undesirable in our case because x must be binary integers. Therefore, we use a quantization technique in the following algorithm for finding the optimal assignment.

Algorithm 1.

1) We start with picking up one arbitrary x_i and set it 0.
2) Afterwards, we solve (28) (29) by convex optimization.
3) However, in most cases, other elements of x will still remain non-binary. Therefore, we choose x_c whose value is closest to 0 or 1, then set it 0 or 1 whichever x_c is closer to.
4) Repeat 2) - 3) until no more non-binary element exists in x.

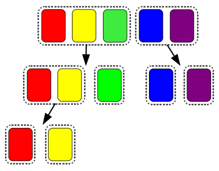


Fig. 4. Example of splitting servers binarily

B. General Solution

Thus far, we have described the basic idea of our relaxed convex optimization approach for the two-server scenario. We can achieve the nearly optimal client-server assignment for M servers by splitting M servers into two groups and recursively splitting within each group as shown in Fig. 4.

How to split M servers is the central question. If M is even, then it makes sense to split M servers into two equal groups with $M/2$ servers in each group. In the ideal case, optimizing the load balance between these two groups will result in individual servers in these two groups having identical server loads. On the other hand, when making the number of servers in these groups is not same, optimizing the objectives in (28) or (29) will result in two groups having total identical server loads. However, since the two groups have different number of servers, a server within a group with fewer servers will likely to have a higher load than a server in the group with more servers. This reduces the load balance. Therefore, when M is not even, it is necessary to modify the objective at each step, depending on how splitting is done, so as to maintain the similar load at individual servers. Intuitively, the modified objective should reflect the number of servers in each group.

Claim 4.1: When splitting M servers into two groups consisting of m and $M-m$ servers in each group, the load balance metrics in (26) and (27) should be replaced by:

$$F^B_{ld} = \tfrac{M^2}{4(M-m)^2}\{\tfrac{2(M-m)}{M} - d^T x\}^2, \quad (30)$$

and

$$F^B_{le} = l'_1 \log_2 l'_1 + l'_2 \log_2 l'_2, \quad (31)$$

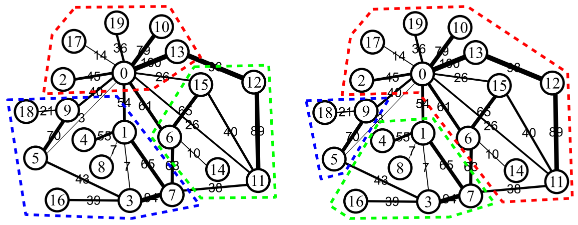
where

$$l'_1 = \{\tfrac{d^T(\mathbf{1}-x)}{2} - \tfrac{2m-M}{M}\}/\tfrac{2(M-m)}{M} \quad (32)$$
$$l'_2 = \tfrac{d^T x}{2}/\tfrac{2(M-m)}{M} \quad (33)$$

Due to lack of space, the justification for this claim is omitted. The full justification can be found in [11]. We would like to mention that the modified load balance metrics allocate a higher load to groups with more servers, and is intuitively plausible. Importantly, both metrics in Claim 4.1 are convex functions, therefore we can employ the Algorithm 1 embedded in the Algorithm 2 below for finding an approximate solution.

Algorithm 2.

1) Split the number of servers into two groups with $m = \lceil \frac{M}{2} \rceil$ and $M - m = \lfloor \frac{M}{2} \rfloor$ servers in each group.
2) Run Algorithm 1 with modified load balance metrics stated in Claim 4.1.
3) Repeat steps 1 and 2 for each of the two groups, until the number of servers in each group equal to 1.



(a) Optimal, BCO-D, BCO-E ($F = 0.183$)

(b) NC ($F = 0.326$)

Fig. 5. Examples of clustering with twenty clients and three servers

C. Time Complexity

Our algorithms run the convex optimization and quantization for N clients and repeat it for $\log M$ depth degrees. Suppose the convex optimization routine takes $f(N)$, then the overall time complexity is $O(N(N + f(N)) \log M)$. $f(N)$ depends on the optimization algorithm but takes at least $O(N)$. As a result, the time complexity of our algorithms is at least $O(N^2 \log M)$, which is considerably expensive.

Though standard calculation of eigenvectors takes $O(N^3)$ in the Normalized Cuts, it reduces the time complexity to $O(N)$ by 1) utilizing sparse graphs 2) calculating only the top of few eigenvectors 3) lowering the precision. Our algorithms have much room for improvement, and speeding up remains as our future work.

V. Simulation Results

In this section, the algorithm names are abbreviated as follows:

BCO-D:	The binary splitting via relaxed convex optimization based on (28)
BCO-E:	The binary splitting via relaxed convex optimization based on (29)
NC:	The Normalized Cuts
Random:	We assign the clients to each server at random.

We used $\beta = 0.5$ for (28) (29).

A. Examples of Graph Clustering

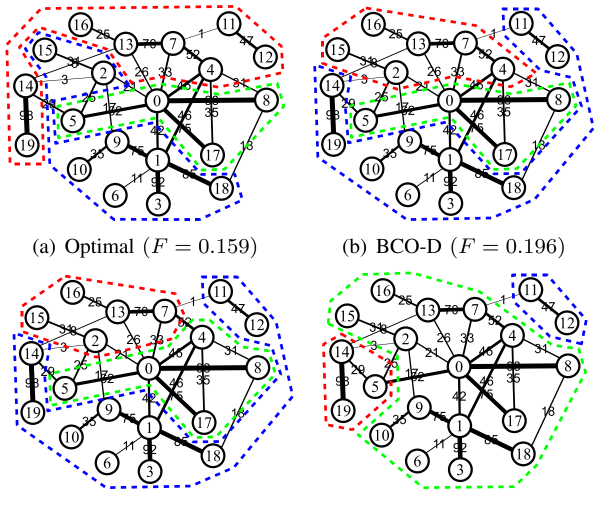
Fig. 5-6 show our simulation results with small graphs. For the comparison, we added the results by the NC. We used a MATALB code [12] written by an author of [3] et al. for the NC algorithm. F is the metric in (17), and the smaller, the better client-server assignment we have.

As shown in the figures, the NC tends to isolate small volumes of groups that do not have strong connection to others, while with our BCO methods, the the volumes of the groups are well balanced, which appears as smaller F values.

B. Optimality

To verify how close the outputs by our algorithms are to the optimal solutions, we made the following examination:

1) For a each graph, we calculate Fs (17) exhaustively for all M^N combinations of X. Herein, we suppose the best (smallest) $F = F_{best}$ and the worst (greatest) $F = F_{worst}$.
2) For each graph, calculate F by each of the BCO-D, BCO-E, NC and a randomly generated X, then calculate F's optimality $\frac{F - F_{worst}}{F_{best} - F_{worst}}$. We also obtain each F's ranking (F_R) out of M^N outputs, and calculate its optimality $1 - \frac{F_R - 1}{M^N - 1}$. The greater those values are, the better optimality.
3) Do 1)-2) for:
   - A hundred different graphs generated by Barabasi-Albert power-law graph generator algorithm.
   - A hundred different graphs generated by our random graph generator. In our random graph generator, each vertex is allocated at most ten randomly selected neighbors.
   - Regular graphs in which every node has the equal number of neighbors (H) with an equal edge weight. Note H is even and vertex i is connected to $i + 1, \ldots, i + \frac{H}{2}, i - 1, \ldots, i - \frac{H}{2}$. We simulated for $H = 2, 4, \ldots, N-2$, that is, $\frac{N}{2} - 1$ different regular graphs for a given $\{M, N\}$.
4) Do 1)-3) for $\{M, N\} = \{2, 30\}, \{3, 20\}$



(a) Optimal ($F = 0.159$)

(b) BCO-D ($F = 0.196$)

(c) BCO-E ($F = 0.235$)

(d) NC ($F = 0.421$)

Fig. 6. Examples of clustering with twenty clients and three servers

$M = 2, N = 30$

	BCO-D	BCO-E	NC	Random
Power-law	93.40%	92.45%	53.57%	42.06%
Random	94.61%	93.90%	62.32%	52.70%
Regular	100.00%	100.00%	96.32%	64.73%

$M = 3, N = 20$

	BCO-D	BCO-E	NC	Random
Power-law	95.14%	94.69%	79.83%	47.36%
Random	93.63%	91.96%	77.40%	50.71%
Regular	99.28%	96.79%	97.99%	40.84%

TABLE II
Optimality of F: $\frac{F - F_{worst}}{F_{best} - F_{worst}}$

$M = 2, N = 30$

	BCO-D	BCO-E	NC	Random
Power-law	99.978%	99.976%	60.73%	49.99%
Random	99.998%	99.993%	66.02%	51.85%
Regular	100.00%	100.00%	99.80%	72.61%

$M = 3, N = 20$

	BCO-D	BCO-E	NC	Random
Power-law	99.998%	99.996%	98.45%	50.58%
Random	99.996%	99.986%	94.73%	50.44%
Regular	99.999%	97.62%	97.65%	18.21%

TABLE III
OPTIMALITY OF F_R (F'S RANKING): $1 - \frac{F_R - 1}{M^N - 1}$

$M = 4, N = 1000$

	BCO-D	BCO-E	NC	Random
F	0.0918	0.0927	0.4943	0.4016
F_c	0.1605	0.1601	0.0063	0.7496
F_l	0.0230	0.0253	0.9824	0.0535
$\frac{l_{max}}{l_{min}}$	1.09	1.10	1032.71	1.22

$M = 7, N = 1000$

	BCO-D	BCO-E	NC	Random
F	0.1169	0.1188	0.4882	0.4698
F_c	0.1962	0.1957	0.0278	0.8574
F_l	0.0375	0.0419	0.9485	0.0821
$\frac{l_{max}}{l_{min}}$	1.20	1.22	1280.21	1.48

$M = 10, N = 1000$

	BCO-D	BCO-E	NC	Random
F	0.1306	0.1318	0.4815	0.4986
F_c	0.2163	0.2146	0.0631	0.9002
F_l	0.0449	0.0491	0.8998	0.0969
$\frac{l_{max}}{l_{min}}$	1.28	1.31	1316.32	1.72

TABLE IV
RESULTS FOR $N = 1000$

Table II and III show average $\frac{F - F_{worst}}{F_{best} - F_{worst}}$ and $1 - \frac{F_R - 1}{M^N - 1}$ values respectively. Overall, the BCO-D method shows the best optimality in spite of the simple and cheap quantization technique (see Section IV-A). In fact, unlike the NC, the BCOs constantly output good F values (close or equal to F_{best}) regardless of the graph type. The simulation shows that the BCO-D is most suitable for solving our problem.

On the other hand, the NC tends to isolate vertices that do not have strong connection to others. This is typically observed in power-law graphs. As a result, though we have low F_c (13) (the amount of inter-server communication), F_l (15) (the load balance metric) becomes high and we have worse F than those of our algorithms.

C. Experiments for Larger Power-law Graphs

As described in Section I-A, our algorithms should perform better than NC for power-law graphs. We simulated for a hundred power-law graphs, in which each vertex is connected to up to a hundred neighbors, with $M = 4, 7, 10, N = 1000$. In this setting, we cannot find the rankings for each algorithm as it requires an exhaustive search over all possible assignments which is infeasible for large N. Instead, Table IV shows the average F (17), F_c (13), F_l (15) values and $\frac{l_{max}}{l_{min}}$ where l_{max}, l_{min} are the maximum and minimum elements in (8) i.e., the maximum and minimum communication load in M servers respectively. As shown by F_l and $\frac{l_{max}}{l_{min}}$ values, the BCO-D and BCO-E fairly balance the load, and at the same time maintain low total communication load as seen in F_cs. Though the NC yields low F_cs, it does not balance the load, which appears as high (bad) F_ls and Fs consequently. Also, the BCOs reduce the overall communication load ($= 1 + F_c$) by 33-36% compared to the random assignment.

VI. CONCLUSION

In this paper, we present a mathematical model and an algorithmic solution to the client-server assignment problem for optimizing the performance of a class of distributed systems over the Internet. We show that in general, finding the optimal client-server assignment for some pre-specified requirements on total load and load balancing is NP-hard, and propose a heuristic via relaxed convex optimization for finding the approximate solution to the client-server assignment problem. Our simulation results indicate that the proposed algorithm almost always finds the optimal solution. Furthermore, the proposed algorithm outperforms other heuristics, including the popular Normalized Cuts algorithm.

REFERENCES

[1] [Online]. Available: http://xmpp.org/about/

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[3] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[4] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1101–1113, August 2002.

[5] K. Lang, "Fixing two weaknesses of the Spectral Method," in *Advances in Neural Information Processing Systems 18*, 2006.

[6] ——, "Finding good nearly balanced cuts in power law graphs," Yahoo Research Labs, Tech. Rep., 2004.

[7] M. Kurucz, A. Benczur, K. Csalogany, and L. Lukacs, "Spectral clustering in telephone call graphs," in *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, ser. WebKDD/SNA-KDD '07. New York, NY, USA: ACM, 2007, pp. 82–91.

[8] J. C. S. Lui and M. F. Chan, "An efficient partitioning algorithm for distributed virtual environment systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, pp. 193–211, March 2002.

[9] P. Morillo, J. M. Orduna, M. Fernandez, and J. Duato, "Improving the performance of distributed virtual environment systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 637–649, 2005.

[10] Y. Deng and R. W. H. Lau, "Heat diffusion based dynamic load balancing for distributed virtual environments," in *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '10. New York, NY, USA: ACM, 2010, pp. 203–210.

[11] H. Nishida, "Optimal client-server assignment for internet distributed system," Oregon State University, Tech. Rep., 2011.

[12] "Ncut." [Online]. Available: http://www.seas.upenn.edu/ timothee/software/ncut/ncut.html

[13] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, March 2004.

[14] "A tutorial on integer programming," The Operations Research Faculty of GSIA, 1997. [Online]. Available: http://mat.gsia.cmu.edu/orclass/integer/integer.html

[15] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001, pp. 849–856.