Optimal Client-Server Assignment for Internet Distributed Systems

Hiroshi Nishida, *Member, IEEE,* Thinh Nguyen, *Member, IEEE,*

***Abstract*—We investigate an underlying mathematical model and algorithms 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. We show that 1) the total communication load and load balancing are two opposing metrics, and consequently, their trade-off is inherent in 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. Our simulation results indicate that the proposed algorithm produces superior performance than other heuristics, including the popular Normalized Cuts algorithm.**

***Index Terms*—Distributed systems, Client-server systems, Graph clustering, Load balancing, Communication overhead, Optimization**

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 the email from its 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 are likely to 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). An 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 such as Google Talk, 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 the 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* $\rightarrow$ *domain* server $\rightarrow$ *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* $\rightarrow$ *domain1* server $\rightarrow$ *domain2* server $\rightarrow$ *bbb*. This design is indeed simple and scalable. If the number of users increases, another server can be added to accommodate the new users.

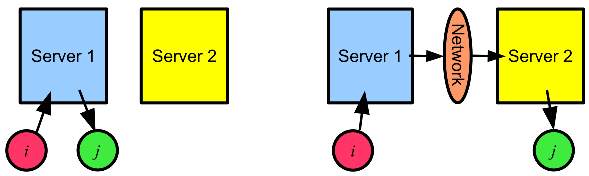
Herein, we consider server load in an IMS. We assume all communications are encrypted. The amount of load on a server (we call it *communication load*) is substantially proportional to the amount of data that the server receives ($= r$) for the following reasons:

- The server basically sends the same amount of data ($= r$) to a client/another server.
- The processing times taken for decrypting the received data and for encrypting the sending data are both proportional to r.
- Except for the encryption and decryption, the load on the server is dominated by copying the data among a network device, the operating system's kernel, an IMS program and sometimes a hard drive, which is also proportional to r.

Based on these, we need to consider how to optimally assign clients to servers, beginning with the following observations:

Nishida is with ASUSA Corporation, Salem OR 97301 USA, and Nguyen is with School of EECS, Oregon State University, Corvallis OR 97331-5501 USA

Email: nishida@asusa.net, thinhq@eecs.oregonstate.edu



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

(b) Client i is assigned to server 1 and client j is assigned to server 2. The messages are passed through server 1 and 2, which doubles the overall communication load.

Fig. 1. Example of client assignment to servers

- Suppose both client i and j are assigned to server 1 and i sends a message of size 1 to j, then the message is sent only via server 1 (see Fig. 1(a)). We define the amount of communication load on server 1 in this case as 1.
- Suppose client i is assigned to server 1 and j is assigned to server 2. If i sends a message of size 1 to j, then the message is delivered through server 1 and 2 (see Fig. 1(b)). The amount of communication load on server 1 is still 1 and that on server 2 is also 1, because both server 1 and 2 need to process the message of size 1. (Note we assume a system always consists of homogeneous machines in this paper.)
- From the above two cases, we know that assigning clients to different servers doubles the amount of total communication load compared to assigning them to the same server. Hence, we need to assign clients to servers so that the amount of total communication load is minimized.
- If two clients who exchange many messages with each other are assigned to two different servers, then the amount of total communication load increases. On the other hand, if two clients who never exchange messages are assigned to different servers, then the amount of total communication load stay unchanged. So, it makes sense to assign clients that exchange many messages to the same server and to assign clients that exchange few messages to different servers in terms of minimizing the overall communication load.
- Since we use multiple servers, we also need to balance the communication load among the servers for the following reasons:
  - As a heavily loaded server typically exhibits a low performance, we would like to avoid the situation.
  - If one server is overloaded, we need to add another server to distribute the load, which is economically inefficient and usually increases the overall communication load (see above). For instance, if the loads on server 1 and 2 are 1.2 (i.e., 20% overloaded) and 0.6 respectively, then we have to add a server to reduce the load on server 1 to less than 1.0. However, if the loads are 0.9 and 0.9, then there no need to do that.
- To minimize the amount of total communication load, assigning all clients to one server is optimal. However, it is impossible due to overloading and completely loses the load balance. Simple load balancing does not usually take account of reducing the overall communication load.

Given the observations above, we must strike a balance between reducing the overall communication load 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 and will result in reducing the overall communication load. 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 contents that are mostly not of interest the user, then on average, every item search by a user will generate a larger communication overhead, as the search must be done across multiple servers. Therefore, letting a user log in the server that likely to have content of interest to a user will raise the efficiency. In this case, the types of contents can also be viewed 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 IMS.

Note we are not focused on real-time (or highly dynamic) client-server assignment in this paper because of the relatively expensive computation cost. Instead, moderately dynamic applications such as social networks where users do not change their friends too frequently are our targets. We assume that our algorithms are used in a such situation that the recalculation of assignment is needed only periodically e.g., once a week.

II. Related Work

A. Clustering Algorithms

To a certain extent, the client-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

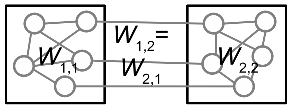
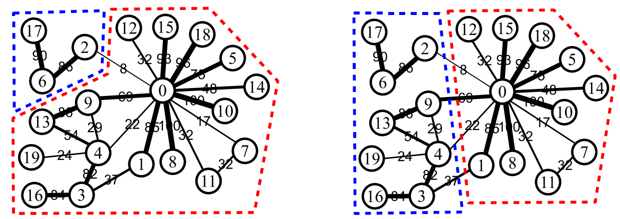


Fig. 2. Example of bi-partitioning



(a) Cut by Normalized Cuts: the total weight of the associated edges are not balanced.

(b) More desirable cut by our algorithm

Fig. 3. Example of clustering

clients, an edge between two vertices denote a communication between two corresponding clients. The weight of an edge between two vertices represents how frequently 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 *Normalized Cuts* (NC) [3]. The NC divides an undirected graph into two disjoint partitions by minimizing

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

where $W_{i,j}$ is the sum of the weights of all edges that connect the vertices in group i and j as shown in Fig. 2. Let F_c denote a metric for the sum of the weights of the inter-group edges and F_l denote a metric for the balance of the sums of the weights of the associated edges in the groups. Suppose those metrics are optimal when they are minimal, then in the above bi-partitioning, F_{ncut} is roughly expressed as $F_c \times F_l$, that is, the less the sum of the weights of the inter-group edges ($W_{1,2} = W_{2,1}$ in Fig. 2) and the more balanced the sums of the weights of the associated edges of the groups ($W_{1,1}+W_{1,2}$ for group 1, $W_{2,2} + W_{2,1}$ for group 2, for the convenience we call *the total weight of associated edges* in this paper) are, the less F_{ncut} we have. The sum of the weights of the inter-group edges corresponds to the amount of the inter-server communication, and the total weight of the associated edges corresponds to the communication load on a server in our problem. Therefore, F_{ncut} is very similar to our objective, though ours is based on $F_c + F_l$ (see Section III-C and III-D).

The NC utilizes the eigenvectors of the adjacency matrix of the graph and provides adroit solution for minimizing (1). The NC is especially suitable for segmenting an image, and is also widely used in bioinformatics and machine learning communities. Different from other methods such as the simple Min cut that only minimizes F_c or [4] that minimizes the largest inter-group flow, the NC indeed considers balancing the total weight of the associated edges of each group. However, it still tends to isolate vertices which do not have strong connection to others and causes unbalance in the total weight of the associated edges of the groups, especially in the power-law graphs [5] [6] [7]. For instance, in Fig. 3(a), the NC clusters only three vertices as a group, and as a result we have the communication load ratio ($= W_{1,1} + W_{1,2} : W_{2,2} + W_{2,1}$, see Section III-B for the details) $1 : 6.5$. Such a solution is not acceptable in practice. A better partition is as shown in Fig. 3(b) whose communication load ratio is $1 : 1.26$. Since the prime applications of our problem are presumed to exhibit power-law graph characteristics, we cannot apply the NC to our problem. In addition, when clustering into $M > 2$ groups, the NC minimizes

$$F_{ncut} = \sum_{i=1}^{M} \frac{\sum_{j\neq i} W_{i,j}}{W_{i,i}+\sum_{j\neq i} W_{i,j}}, \quad (2)$$

and it is not sure whether minimizing (2) will always minimize our objective.

Deng et al. [8] introduce an efficient graph clustering algorithm called *Graclus* which utilizes the equivalence between kernel k-means [9] and other graph clustering algorithms including the NC. The Graclus eliminates the time-consuming calculation of eigenvectors inherent in the NC, and minimizes the objective (2) faster than the NC. Moreover, similarly to Metis [10], its three-step ‘coarsening-base clustering-refining’ multilevel process enables more ‘balanced’ clustering, and as a result obtains better (smaller) objective values than the NC. However, as well as the other works [10] [6] [11] [12] [13], the ‘balanced’ clustering herein means balancing each group *size*, i.e., *the number of vertices* in a group. In our problem, balancing the total weight of the associated edges of each group is required, and therefore, the results of the Graclus etc. are not directly applicable to our problem.

Lang [6] examines 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. However, similar to the Graclus, it focuses on balancing each group size.

Other representative spectral clustering methods *Ratio Cut* [14] and *Min-max Cut* [15] balance the size and the *volume* (= *the sum of the weights of all edges in a group*, equivalent to $W_{i,i}$ in Fig. 2) of each group respectively, and therefore are less relevant to our research than the NC.

To the best of knowledge, there is no clustering algorithm which achieves our goal: minimizing $F_c + F_l$.

B. Load Balancing in Distributed Systems

In classical task assignment problems in distributed systems, such as those stated in [16] and Chapter 7.3 of [17], the optimal assignment is pursued for given execution cost (load) of each task and inter-task communication cost. In this model, the execution cost is assumed to be invariable regardless of the task assignment. This is the typical and most common premise for traditional load balancing in distributed systems, and is inapplicable to our problem. In our model, the amounts of tasks (loads) dynamically vary according to the task assignment. To the best of our knowledge, there are no researches that focus on load balancing in the same model.

In the recent research, the client-server assignment for distributed virtual environment (DVE) systems exhibits a similar set of issues: balancing the workload and reducing the communication between the servers. The DVE systems allow multiple users working on different client computers to interact in a shared virtual world. For example, [18] [19] [20] [21] study efficient client-server assignment for DVE systems. Especially, [21] takes account of extra inter-server communication caused by different client-server assignments. However, the load by communication is not seriously considered in DVE systems since the load is primarily generated by processing 3-D images. Therefore, unlike our problem, their overall workload is assumed to be constant regardless of the client-server assignment, and [21] uses the amount of communication as a constraint for their optimization problem (Eq. (10)).

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 notation.

A. Notation

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

- $|v|_1$: Norm-1 of vector v, i.e., the sum of all elements in v.
- $\|A\|_1$: 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 $\|S\|_1 = 1$ 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, $\sum_{s=1}^{M} X_{i,s} = 1\ \forall i$.

Next, we will derive the expressions for the communication load, i.e., the amount of communication data 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:

$$P_{s,t} = \sum_{i=1}^{N}\sum_{j=1}^{N} S_{i,j} X_{i,s} X_{j,t}, \tag{3}$$

that is:

$$P = X^T S X, \tag{4}$$

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 a message is passed between two clients only through a single server, i.e., the two clients are assigned to the same server, the amount of communication load on the server is $1 \times$ {the size of the messages}. However, when the two clients are assigned to different servers, the amount of communication load is $1 \times$ {the size of the messages} for each server and $2 \times$ {the size of the messages} in total. Thus, to calculate the communication load, two different types of message passing need to be considered:

1) Message passing through a single server, i.e., intra-server communication.
2) Message passing through two servers, i.e., 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 data processing. As a result, let $L \in [0,1]^{M\times M}$ represent the load generated by the message exchanges, then

$$L = P + P^T - P^D, \tag{5}$$

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 $P = X^T S X$, $P^T = (X^T S X)^T = X^T S^T X$, $P^D = (P + P^T)^D/2 = (X^T S X + X^T S^T X)^D/2$, we have:

$$L = X^T S X + X^T S^T X - \frac{1}{2}(X^T S X + X^T S^T X)^D. \tag{6}$$

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

$$Q = X^T A X \tag{7}$$

$$L = Q - \frac{1}{2}Q^D. \tag{8}$$

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

$$l = L\mathbf{1}, \tag{9}$$

where $\mathbf{1}$ denotes a column vector whose all elements are 1, and l_i is in other words the total weight of associated edges of group i (see Section II-A).

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:

$$\|L\|_1 = \|Q - \tfrac{1}{2}Q^D\|_1 = \|\tfrac{1}{2}Q + \tfrac{1}{2}(Q - Q^D)\|_1 \tag{10}$$

$$= 1 + \|\tfrac{1}{2}(Q - Q^D)\|_1 = 1 + \textstyle\sum_{s=1}^{M}\sum_{t=1}^{s-1} L_{s,t}. \tag{11}$$

Let

$$F_c = \sum_{s=1}^{M}\sum_{t=1}^{s-1} L_{s,t}, \tag{12}$$

then F_c is 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 1 plus the amount of the inter-server communication (F_c), where the amount of the inter-server communication can be expressed as the extra load caused by distributing the servers; if all clients are assigned to a single server, the total communication load is 1. From the above equation, we can regard F_c as the metric for the total communication load. Note $0 \leq F_c \leq 1$, and 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 the 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 communication load, the Gini coefficient is defined as:

$$F_l = \tfrac{M}{M-1}\left(\tfrac{2\sum_{s=1}^{M} s\, l_s}{M\sum_{s=1}^{M} l_s} - \tfrac{M+1}{M}\right) \tag{13}$$

$$= \tfrac{1}{M-1}\left(\tfrac{2\sum_{s=1}^{M} s\, l_s}{\sum_{s=1}^{M} l_s} - M - 1\right), \tag{14}$$

where $l_1 \leq l_2 \leq \cdots \leq l_M$. 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 (14) = 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

$$F = \alpha F_c + (1-\alpha)F_l, \tag{15}$$

where $0 \leq \alpha \leq 1$ is an arbitrary coefficient. We want to minimize F. Note that $0 \leq F \leq 1$, 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:

$$F = 0.5F_c + 0.5F_l. \tag{16}$$

As mentioned in Section II-A, $F_c \times F_l$ also shows similar characteristics to F: the smaller F_c and F_l are, the smaller $F_c \times F_l$ we get. However, for $0 \leq F_c, F_l \leq 1$, if F_c (or F_l) = 0, then we have optimal $F_c \times F_l$ $(=0)$ regardless of F_l (or F_c) value. This is not desirable and will produce unbalanced clustering as shown in Fig. 3. Hence, we employ $F_c + F_l$ style for our research.

As a consequence, our optimization problem is formally cast as:

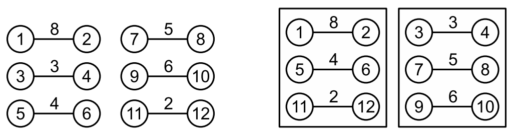
$$\begin{array}{ll} \text{Minimize} & F \\ \text{Subject to} & X \in \{0,1\}^{N\times M},\ \sum_{j=1}^{M} X_{i,j} = 1\ \forall i. \end{array} \tag{17}$$

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 (17) is NP-hard.

Proof: The main idea is to show that the well-known *partition problem* is a special case of our problem. The partition problem is to decide whether a given set of integers can be partitioned into two sets with identical sums, and is known to be NP-complete. Suppose our problem is to decide if there is an assignment s.t. $F = 0$ (i.e., $F_c = F_l = 0$) for $M = 2$, that is, there is no communication between the two servers and the loads are completely balanced. For a given set of integers in the partition problem, we can always construct a corresponding special graph for our problem in polynomial time that represents the communication pattern of the clients s.t. 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 a graph with $N = 2K$ clients s.t. each client is to



(a) Example of graph (b) Optimal clustering

Fig. 4. Special case of our problem is equivalent to partition problem

communicate with exactly one other client. Therefore, there are 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. This mapping takes $O(K)$ steps.

For example, suppose the given set of integers for the partitioning problem is $\{8, 3, 4, 5, 6, 2\}$, then we construct a graph $G = (V, E)$ s.t. $V = \{v_1, \ldots, v_{12}\}$, $E = \{e_1, \ldots, e_6\}$, $e_1 = (v_1, v_2)$, $e_2 = (v_3, v_4)$, $\ldots$, $e_6 = (v_{11}, v_{12})$ and assign the weights $w(e_1) = 8$, $w(e_2) = 3$, $\ldots$, $w(e_6) = 2$ as seen in Fig. 4(a). Since $\{8, 3, 4, 5, 6, 2\}$ can be partitioned into $\{8, 4, 2\}$ and $\{3, 5, 6\}$ so that the sums of the subsets are equal, our problem can also obtain the optimal assignment with $F = 0$ as illustrated in Fig. 4(b). Clearly, the original partition problem has an answer *yes* iff our problem has an answer *yes*.

Hence, our problem is NP-hard since the partition problem is NP-complete. ■

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 communication are:

$$L^B = \begin{pmatrix} \frac{1}{2}(\mathbf{1}-x)^T A(\mathbf{1}-x) & (\mathbf{1}-x)^T Ax \\ x^T A(\mathbf{1}-x) & \frac{1}{2}x^T Ax \end{pmatrix}, \qquad (18)$$

where $\mathbf{1}$ is a vector with N ones, A is a matrix from (7), $L^B_{1,1} = \frac{1}{2}(\mathbf{1}-x)^T A(\mathbf{1}-x)$ is the amount of communication 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 communication exchanged between server 1 and 2, $L^B_{2,2} = \frac{1}{2}x^T Ax$ is the amount of communication 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:

$$F^B_c = x^T(D-A)x, \qquad (19)$$

which is equivalent to F_c (12) 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,

$$\tfrac{1}{2}(\mathbf{1}-x)^T A(\mathbf{1}-x) = \tfrac{1}{2}\{d^T(\mathbf{1}-x) - F^B_c\} \qquad (20)$$

$$\tfrac{1}{2}x^T Ax = \tfrac{1}{2}(d^T x - F^B_c), \qquad (21)$$

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

$$L^B = \begin{pmatrix} \frac{1}{2}\{d^T(\mathbf{1}-x) - F^B_c\} & F^B_c \\ F^B_c & \frac{1}{2}(d^T x - F^B_c) \end{pmatrix}, \qquad (22)$$

and the communication loads are:

$$l_1 = L^B_{1,1} + L^B_{1,2} = \tfrac{1}{2}\{d^T(\mathbf{1}-x) + F^B_c\}, \qquad (23)$$

$$l_2 = L^B_{2,1} + L^B_{2,2} = \tfrac{1}{2}(d^T x + F^B_c). \qquad (24)$$

Based on (23) (24), we propose two convex functions that approximate our original non-convex 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.,

$$F^B_{lv} = (1 - d^T x)^2 \qquad (25)$$

can be utilized as a new load balance metric. Note F^B_{lv} is convex and $0 \le F^B_{lv} \le 1$. 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 (25) approximately minimizes the Gini coefficient.

The second convex function is based on the entropy of l_1 and l_2. In (23) (24), 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:

$$F^B_{le} = \tfrac{d^T(\mathbf{1}-x)}{2}\log_2\tfrac{d^T(\mathbf{1}-x)}{2} + \tfrac{d^T x}{2}\log_2\tfrac{d^T x}{2}. \qquad (26)$$

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 we have. In an ideal case, both the negative entropy and the Gini coefficient are minimized when the communication load distribution is uniform. Thus, we approximate the Gini coefficient with the negative entropy function which is convex.

As a result, (19) + (25) and (19) + (26), i.e.,

$$F^B_v = \beta F^B_c + (1-\beta)F^B_{lv} \qquad (27)$$

$$F^B_e = \beta F^B_c + (1-\beta)F^B_{le} \qquad (28)$$

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

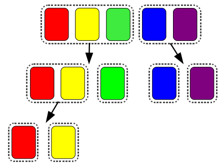


Fig. 5. Example of splitting servers binarily

F_{lv}^B (or F_{le}^B). We test for $\beta = 0.5, 0.3, 0.1$ and 0.05 in the simulation (Section V).

Next, we relax the constraint of x_i being binary, to allow $x_i \in [0, 1]$. However, (27) (28) with a weak constraint such as $0 \leq x \leq 1$ 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 (Two-server Algorithm).

1) We start with picking up one arbitrary x_i and set it 0.
2) Afterwards, we solve (27) or (28) 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.

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 an approximately optimal client-server assignment for M servers by splitting M servers into two groups and recursively splitting within each group as shown in Fig. 5.

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 communication loads. On the other hand, when making the number of servers in these groups is not same, optimizing the objectives in (27) or (28) will result in two groups having total identical communication 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 (25) and (26) should be replaced by:

$$F_{lv}^B = \tfrac{M^2}{4(M-m)^2}\{\tfrac{2(M-m)}{M} - d^T x\}^2, \tag{29}$$

and

$$F_{le}^B = l_1' \log_2 l_1' + l_2' \log_2 l_2', \tag{30}$$

where

$$l_1' = \{\tfrac{d^T(\mathbf{1}-x)}{2} - \tfrac{2m-M}{M}\}/\tfrac{2(M-m)}{M} \tag{31}$$

$$l_2' = \tfrac{d^T x}{2}/\tfrac{2(M-m)}{M}. \tag{32}$$

The justification of these modified objectives can be found in the Appendix. We would like to mention that intuitively, the modified load balance metrics above allocate a higher load to groups with more servers. Importantly, both (29) and (30) 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 (General Algorithm).

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 metric (29) or (30).
3) Repeat 1) – 2) for each of the two groups, until the number of servers in each group equal to 1.

Note in order to obtain a better result, we will also need to run Algorithm 1 twice at step 2) of Algorithm 2 when M is odd, as follows:

1) At first, run Algorithm 1 normally.
2) At second, run Algorithm 1 by setting x_i 1 instead of 0 at step 1).

Then choose a result with smaller F. This is because different results will be obtained by forcing x_i to initially assign to the two different groups since the graph is not symmetrically split. Our simulation in Section V employs this way.

C. Time Complexity

The proposed two-server algorithm consists of solving the N convex optimization problems, each problem corresponds to a quantization of a client (x_i). For each quantization, we need to search over all possible N clients. Suppose the convex optimization routine takes $f(N)$, then the time complexity of the two-server algorithm is $O(N(N + f(N))$. The general algorithm consists of running the two-server algorithm about $\log M$ times for M servers. Thus, 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.

The Normalized Cuts achieves approximately $O(N)$ time complexity by introducing fast eigenvector calculation (including lowering the precision), though normal eigenvector calculation takes $O(N^3)$. Similarly, the time complexity of the Graclus is also about $O(N)$ due to the coarsening-refining scheme and kernel k-means. Since our programs use generic optimization library Ipopt [22], they are currently much slower than the NC and Graclus. Optimizing our techniques remains as future work, and we note that it is not the scope of this paper. However, considering the goal of most graph clustering algorithms is balancing the size of each group, we think that a certain level of high time complexity for our problem is inevitable because balancing the total weights of associated edges is more of the challenge.

V. Simulation Results

In this section, we evaluate the performance of the following algorithms abbreviated as follows:

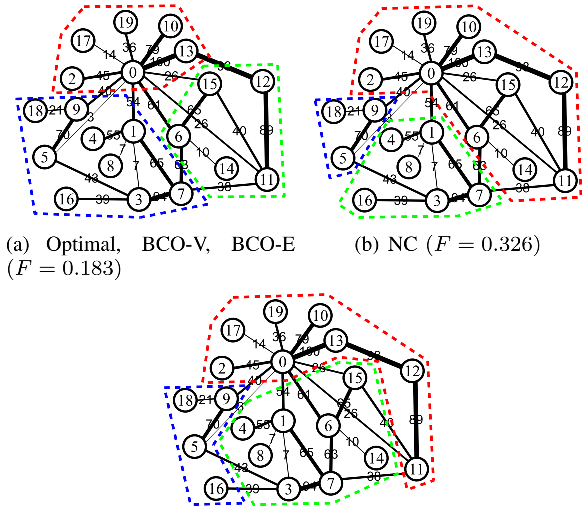
- BCO-V: The binary splitting via relaxed convex optimization based on (27)
- BCO-E: The binary splitting via relaxed convex optimization based on (28)
- NC: The Normalized Cuts
- Graclus: An efficient graph clustering algorithm in [8] (see also Section II-A)
- RND: The random client-server assignment

We examined the BCO-V and BCO-E for $\beta = 0.5$, 0.3, 0.1, 0.05 and used Ipopt [22] as a convex optimization solver. For the NC algorithm, we used the MATLAB code at [23], and for the Graclus, we used the version 1.2 code at [24]. Both programs are written by the authors of their original papers [3] [8]. Note we ran C based programs (BCOs, Graclus) with Intel Pentium E2140 machines and ran MATLAB based programs (NC) with High Performance Computing Cluster at Oregon State Univ. Therefore, the comparison on the computation time is approximate.

A. Examples of Graph Clustering

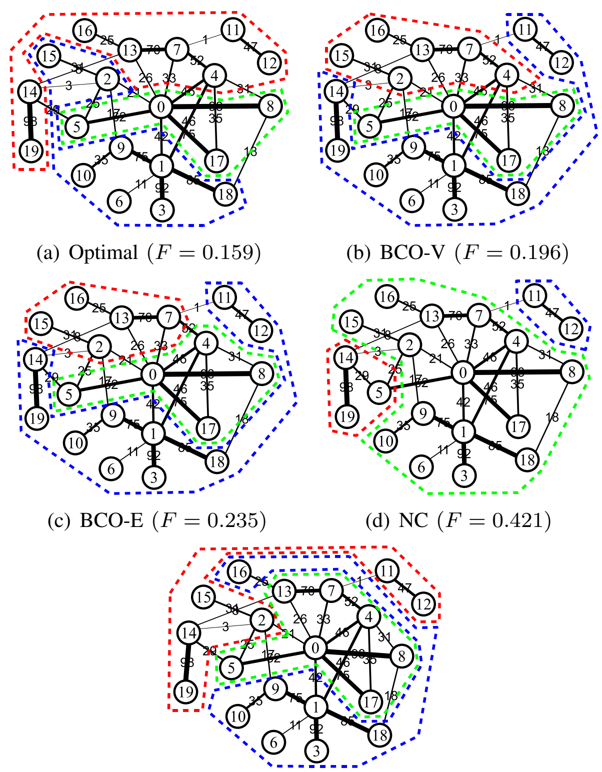
Fig. 6–7 show our simulation results with small graphs. The reason for using a small graphs is because it is easy to examine the results of each algorithms in details. Furthermore, it is feasible to find the optimal solution by an exhaustive search, which helps us to quantify how good an approximate solution produced by a heuristic is. In the simulation, $\beta = 0.5$ is used for the BCOs and $\rho = 1$ is used for the ALSs. Also, the results by the NC and Graclus are added for the comparison. F is the metric in (16), 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. Our BCO methods well balance the total weight of the associated edges in each group, which appears as smaller F values. The Graclus clusters better than the NC but does not better than the BCOs.



(a) Optimal, BCO-V, BCO-E ($F = 0.183$)
(b) NC ($F = 0.326$)
(c) Graclus ($F = 0.297$)

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



(a) Optimal ($F = 0.159$)
(b) BCO-V ($F = 0.196$)
(c) BCO-E ($F = 0.235$)
(d) NC ($F = 0.421$)
(e) Graclus ($F = 0.252$)

Fig. 7. Examples of clustering with twenty clients and three servers

B. Optimality

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

1) For each graph, we calculate Fs (16) exhaustively for all M^N combinations of X. Herein, we suppose the best (smallest) $F = F_{best}$ and the worst (largest) $F = F_{worst}$.
2) For each graph, calculate F by each of the BCO-V, BCO-E, NC, Graclus and a randomly generated X (RND), 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 then calculate its optimality $1 - \frac{F_R - 1}{M^N - 1}$. The larger those values are, the better optimality we have.

3) Do 1) – 2) for:
   - A hundred random 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 $\{M, N\} = \{2, 30\}, \{3, 20\}$.

Table I and II show average $\frac{F - F_{worst}}{F_{best} - F_{worst}}$ and $1 - \frac{F_R - 1}{M^N - 1}$ values for a hundred graphs created by each of the power-law, random and regular graph generators respectively. Overall, the BCOs for $\beta = 0.3$ or 0.1 show 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 BCOs are very 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 for $M = 2$, $N = 30$. As a result, though we have small amount of inter-server communication F_c, the load balance metric F_l becomes large and F produced by NC is larger (worse) than those produced by our algorithms.

The Graclus exhibits more balanced cuts than the NC for any type of graph. This is due to its multilevel process; balancing the size of each group is somehow related to balancing the weights of associated edges in small graphs. As a result, the Graclus achieves smaller F_l and F than the NC. However, it does not necessarily result in balancing the total weight of associated edges of each group, and therefore the Graclus does not perform better than the BCOs.

As for the computation time, all the methods finish clustering a graph literally in a moment (≤ 1 second).

C. Experiments for Larger Power-law Graphs

As described in Section I-A, our algorithms should perform better than the NC and Graclus 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 $M = 4, 7, 10$ and $N = 1000$. 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 III shows the average F (16), F_c (12), F_l (14) values and $\frac{l_{max}}{l_{min}}$ where l_{max}, l_{min} are the maximum and minimum elements in (9) i.e., the maximum and minimum communication load in M servers respectively. $\frac{l_{max}}{l_{min}}$ is another side metric that reflects the load balance among the servers.

As shown by the F_l and $\frac{l_{max}}{l_{min}}$ values, the BCO-V and BCO-E fairly balance the load, and at the same time maintain low total communication load as seen in their F_cs. Though the NC yields low F_cs, it does not balance the load, which appears as large F_l, F and $\frac{l_{max}}{l_{min}}$ values consequently. The Graclus performs more balanced cuts than the NC, but its $\frac{l_{max}}{l_{min}}$ values are higher than 2, which will not be acceptable in real distributed systems. Interestingly, the F_c values of the Graclus are very close to those of the BCOs. Hence, the differences in their F values are mainly determined by their F_l values; F_ls of the Graclus are higher than those of the BCOs. This also substantiates that our algorithms aptly strikes the balance between the two opposing metrics: reducing the total communication load and load balance.

Also, the BCOs reduce the total communication load ($= 1 + F_c$, see (11) (12)) by 33-36% compared to the random assignment. This indicates that a system that uses 100 servers with a random client-server assignment requires only 64-67 servers (or less because the inter-server communication will also decrease by reducing the number of servers) with an assignment by the BCOs. Also, since $1 \leq 1 + F_c \leq 2$ (see (11) and Section III-C), the maximum reduction rate of the total communication load $= 50\%$ $(= 1/2)$. Thus, the reduction rates of 33-36% are significantly high, and we can also know how inefficient the random assignment is, though it yields 'not bad' load balance (see their $\frac{l_{max}}{l_{min}}$ values). This also verifies the effectiveness of our algorithms.

As for the computation time taken for clustering a graph, the Graclus and NC finish within a second, while the BCOs take 18-58 minutes. The BCOs for $N = 7$ take longer than those for $N = 10$. This is because as described in the end of Section IV-B, we examine binary split twice for odd M, and therefore the initial split for $M = 7$ takes longer than that for $M = 10$. We recognize the expensiveness of our algorithms. The improvement in the time complexity is the next step of our research.

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] The XMPP Standards Foundation, "XMPP," 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.

[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.

TABLE I

OPTIMALITY OF F: $\frac{F-F_{worst}}{F_{best}-F_{worst}}$

$M=2, N=30$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
Power-law	93.40%	94.20%	94.46%	94.39%	92.45%	93.81%	**94.47%**	94.39%	53.57%	82.99%	42.06%
Random	94.61%	96.44%	97.04%	97.08%	93.90%	96.28%	97.05%	**97.10%**	62.32%	82.51%	52.70%
Regular	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	96.32%	**100.00%**	64.73%

$M=3, N=20$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
Power-law	95.23%	95.45%	93.96%	93.49%	94.68%	**95.50%**	94.43%	93.90%	79.82%	87.71%	46.94%
Random	93.63%	**94.86%**	94.57%	94.35%	91.96%	94.80%	94.72%	94.40%	77.40%	80.90%	50.71%
Regular	99.28%	99.53%	99.27%	99.75%	96.79%	**99.76%**	**99.76%**	99.75%	97.99%	99.51%	40.84%

TABLE II

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

$M=2, N=30$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
Power-law	99.978%	**99.979%**	99.970%	99.970%	99.976%	**99.979%**	99.970%	99.970%	60.73%	96.65%	49.99%
Random	99.998%	99.9995%	99.99992%	**99.99993%**	99.993%	99.9994%	**99.99993%**	**99.99993%**	66.02%	97.13%	51.85%
Regular	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	99.80%	**100.00%**	72.61%

$M=3, N=20$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
Power-law	**99.998%**	99.996%	99.97%	99.93%	99.996%	**99.998%**	99.98%	99.96%	98.49%	99.61%	49.64%
Random	99.996%	**99.9993%**	**99.9993%**	99.9989%	99.986%	**99.9993%**	**99.9993%**	99.9989%	94.73%	98.89%	50.44%
Regular	99.9991%	99.9997%	99.92%	99.92%	97.62%	**99.99997%**	99.9994%	99.92%	97.65%	99.9994%	18.40%

TABLE III

RESULTS FOR $N=1000$

$M=4, N=1000$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
$\boldsymbol{F}$	0.0918	0.0892	0.0886	**0.0885**	0.0927	0.0900	0.0888	0.0886	0.4943	0.1794	0.4016
F_c	0.1605	0.1611	0.1624	0.1627	0.1601	0.1610	0.1621	0.1624	**0.0063**	0.1600	0.7496
F_l	0.0230	0.0172	0.0149	**0.0144**	0.0253	0.0190	0.0156	0.0148	0.9824	0.1987	0.0535
$\frac{l_{max}}{l_{min}}$	1.09	1.07	1.06	**1.05**	1.10	1.07	1.06	1.06	1032.71	2.47	1.22
Time	18m36s	18m56s	19m44s	20m16s	18m40s	18m52s	19m32s	20m3s	$\leq$ 1sec	$\leq$ 1sec	-

$M=7, N=1000$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
$\boldsymbol{F}$	0.1169	0.1142	**0.1129**	0.1133	0.1188	0.1155	0.1131	0.1133	0.4882	0.2067	0.4698
F_c	0.1962	0.1971	0.1991	0.2002	0.1957	0.1968	0.1981	0.1998	**0.0278**	0.1977	0.8574
F_l	0.0375	0.0314	0.0267	**0.0264**	0.0419	0.0342	0.0280	0.0267	0.9485	0.2157	0.0821
$\frac{l_{max}}{l_{min}}$	1.20	1.16	**1.13**	1.14	1.22	1.18	1.14	**1.13**	1280.21	3.72	1.48
Time	50m58s	53m4s	56m16s	57m50s	48m13s	50m	53m12s	56m50s	$\leq$ 1sec	$\leq$ 1sec	-

$M=10, N=1000$

	BCO-V				BCO-E				NC	Graclus	RND
	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$	$\beta=0.5$	$\beta=0.3$	$\beta=0.1$	$\beta=0.05$			
$\boldsymbol{F}$	0.1306	0.1279	**0.1267**	0.1278	0.1318	0.1288	0.1269	0.1270	0.4815	0.2202	0.4986
F_c	0.2163	0.2175	0.2195	0.2216	0.2146	0.2168	0.2188	0.2204	**0.0631**	0.2127	0.9002
F_l	0.0449	0.0383	0.0339	0.0340	0.0491	0.0408	0.0350	**0.0337**	0.8998	0.2276	0.0969
$\frac{l_{max}}{l_{min}}$	1.28	1.23	**1.20**	1.21	1.31	1.25	1.21	**1.20**	1316.32	4.35	1.72
Time	26m18s	26m58s	28m25s	29m17s	25m40s	26m8s	27m23s	26m12s	$\leq$ 1sec	$\leq$ 1sec	-

[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] I. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944 –1957, November 2007.

[9] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, pp. 1299–1319, July 1998.

[10] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, pp. 359–392, December 1998.

[11] M. L. Huang and Q. V. Nguyen, "A fast algorithm for balanced graph clustering," in *Proceedings of the 11th International Conference Information Visualization*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 46–52.

[12] K. Andreev and H. Räcke, "Balanced graph partitioning," in *In 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*. ACM Press, 2004, pp. 120–124.

[13] F. Nie, C. Ding, D. Luo, and H. Huang, "Improved MinMax cut graph clustering with nonnegative relaxation," in *Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part II*, ser. ECML PKDD'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 451–466.

[14] P. Chan, M. Schlag, and J. Zien, "Spectral k-way ratio-cut partitioning and clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 9, pp. 1088 –1096, Sep. 1994.

[15] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A Min-max Cut Algorithm for Graph Partitioning and Data Clustering," in *Proceedings of ICDM 2001*, 2001, pp. 107–114.

[16] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. Softw. Eng.*, no. 1, pp. 85–93, 1977.

[17] P. Sinha, *Distributed Operating Systems: Concepts and Design*. IEEE Press, 1997.

[18] J. C. S. Lui and M. F. Chan, "An efficient partitioning algorithm for distributed virtual environment systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 193–211, March 2002.

[19] 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.

[20] 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.

[21] D. N. B. Ta and S. Zhou, "Efficient client-to-server assignments for distributed virtual environments," in *Proceedings of the 20th international conference on Parallel and distributed processing*, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006.

[22] COIN-OR, "Ipopt," https://projects.coin-or.org/Ipopt.

[23] T. Core, "Normalized cuts segmentation code, for matlab," http://www.seas.upenn.edu/˜timothee/software/ncut/ncut.html.

[24] Graclus, "Graclus software," http://www.cs.utexas.edu/users/dml/Software/graclus.html.

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

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

[27] 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.

[28] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 551–556.

[29] W. E. Donath and A. J. Hoffman, "Lower bounds for the partitioning of graphs," *IBM Journal of Research and Developement*, vol. 17, pp. 420–425, 1973.

[30] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 2, pp. 241–254, 1967.

[31] C.-K. Cheng and Y.-C. Wei, "An improved two-way partitioning algorithm with stable performance [vlsi]," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 12, pp. 1502 –1511, Dec. 1991.