An Approximation Algorithm for Client Assignment in Client/Server Systems

Yuqing Zhu*, Weili Wu†*✉ , James Willson‡, Ling Ding§, Lidong Wu*, Deying Li¶, and Wonjun Lee‖
*Department of Computer Science, University of Texas at Dallas. Email: yuqing.zhu@utdallas.edu
†College of Computer Science and Technology, Taiyuan University of Technology, China.
‡Department of Computer Science, University of New Mexico.
§Institute of Technology, University of Washington Tacoma.
¶School of Information, Renmin University of China.
‖Dept. of Computer Science & Engineering, Korea University.

***Abstract*—One type of distributed systems is the client/server system consist of clients and servers. In order to improve the performance of such a system, client assignment strategy plays an important role. There are two criteria to evaluate the load on the servers — total load and load balance. The total load increases when the load balance decreases, vice versa. It has been proved that finding the best client assignment is NP-hard. In this paper, we propose a new model for the client assignment problem and design an algorithm based on Semidefinite programming (SDP). Our method has a (relaxed) performance ratio 0.87 when only 2 servers exist. In general case, our method becomes a heuristic, and the ratio of each iteration is 0.87. We are the first one to give these bounds. Our simulation results are compared with the state-of-art client assignment method, and our strategy outperforms it in terms of running time while keeps the load in similar level.**

I. Introduction

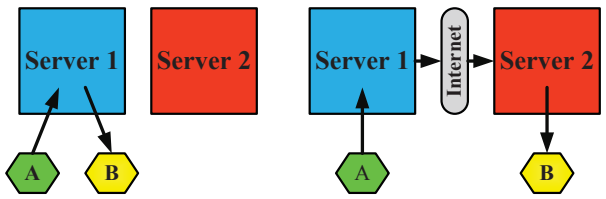
There are two main system architectures of the Internet distributed systems — one is client-server architecture [1] and the other one is peer-to-peer architecture. In this paper, we will study the client-server architecture, which is a network architecture in which each computer or process on the network is either a client or a server. A "client" is a program or a terminal which the user accesses in the first place. Each client communicates with each other through the "servers". The communications among the servers are direct. That is, any server can communicate with another server directly. If one client A wants to send a message to client B, it will firstly send the message to the server which A is assigned to, and this server will directly forward the message to B if B and A are assigned to the same server. If B and A are assigned to different servers from A, then the message will be forwarded from A's assigned server to B's assigned server first and then passed to B. The advantage of this system is that the more powerful servers let the clients free from the responsibility of processing and storing. The system structure is also more scalable. Hence, how to achieve a good client assignment strategy is essential and critical to client/server systems.

The client-server architecture has become one of the basic models of network computing. Many applications such as E-mail exchange, web access and database access, are based on the client-server architecture. A web browser is a client program at the user computer that may access information at any web server in the world. While in some email systems, the assignments depend on the clients' organizations. If two clients work for the same company, they should be assigned to the same email server. Social network applications are also the possible applications. Users can exchange their messages and photos in Facebook. Relational database applications is suited for client-server computing too. There are two components to the client-server database access protocol. One deals with communication between the servers (i.e. TCP/IP), and the other deals with the interaction of the client and the server (i.e. SQL). A client uses SQL to query a machine, if this machine does not have the data this client needs, this machine talks to other machines via TCP/IP to seek the data. MapReduce [2] is also a potential application. Allotting the often queried search keywords together on a same server can decrease the inter-server communication. The search keywords is the clients in this case.

Another example is the Instant Message System (IMS). The IMS servers will forward the messages for the clients. Different from the email system, the client assignments are not constrained by the clients' organization, in an IMS with the XMPP (Jabber) [3] protocol. In XMPP, the format of a user is *user@domain* and the domain usually represents a corresponding server. When the existing resources are not enough, more servers are added to improve the performance and the throughput of the system. For this case, to assign the clients, we should consider the communication load in client assignments.

The communication load in client/server systems can be further divided into total load and load balance. Total load means the load amount on all servers and load balance represents the load difference among the servers. To understand this let us check Fig. 1 which plots two possible cases of assigning two clients A and B to two servers 1 and 2. In Fig.

✉ Corresponding Author: Weili Wu. Email: weiliwu@utdallas.edu
This work was supported in part by NSF grants CNS-1016320 and CCF-0627233, NSFC grants 61070191 and 91124001, and Shenzhen Strategic Emerging Industries Program with Grant No. ZDSY20120613125016389.
978-1-4799-3360-0/14/ $31.00 ©2014 IEEE



(a) A and B connect to the same server

(b) A and B connect to different servers

Figure 1. An example of client assignment.

1(a), both A and B are assigned to server 1. In this case, all the communication from A to B is within server 1, and the message processing is server 1 receiving from A and sending to B. In Fig. 1(b), A is assigned to server 1 and B is assigned to server 2. In this case, the communication from A to B asks for the talk between servers, and the message processing includes: server 1 receives from A and sends to server 2, and server 2 receives from server 1 and sends to B. The load is not balanced in Fig. 1(a) since only one server has messages to process, while in Fig. 1(b) the server loads are balanced but the scatter of clients leads to extra load in the system, which means, the total load, i.e. the sum of server 1's load and server 2's load increases. In general, total load and load balance are opposite. To reduce the total load always makes the loads more unbalanced, i.e. the assignment in Fig. 1(a) performs well on total load minimization but badly on load balancing, while the assignment in Fig. 1(b) performs well on load balancing but badly on total load minimization. Nevertheless, both total load and load balance are significant. If the total load is high, then the system may have taken resource that could have been saved. If the load balance is bad, then the servers with the heaviest loads are too busy but the others are starving. Thus, in this paper, we will study how to assign the clients to servers, aiming to strike a balance between the total communication load amount and the load balance on servers. This problem had been proved to be **NP**-hard in [1].

Our contributions in this papers are as follows:

1) We give a new mathematical model of the client assignment problem, and present algorithm *Binary splitting methods based on solving SDP* (*BSP*). For the special case where there exists only 2 servers, we prove that its ratio is 0.87. For the general case with k servers, we also prove that for each iteration in *BSP*, its performance ratio is 0.87. Although the ratios are not strict ones, they make our approximation algorithm *BSP* the first bounded one for our kind of problem.
2) Comparing to the best known algorithm, *BCO-V* proposed in [1], which asks solving convex optimization problem for $O(n)$ rounds in each iteration where n is the number of clients in that iteration, our algorithm only requires solving SDP once in each iteration. Hence, the running time of *BSP* is an order of magnitude smaller than that of *BCO-V* theoretically.
3) Through simulation experiments, comparing to the state-of-art algorithm, *BSP* performs almost as well on minimizing the objective function, but greatly outperforms it in terms of running time. Hence we evaluate the efficacy of *BSP*.

II. Related Work

Our client assignment problem can be viewed as a clustering problem to some extent. Regard the clients as the nodes in a graph, and the communications between the clients as the edges. Each directed edge from node u to v represents the message rate from client u to v. For any pair of nodes u and v, if they are put into different clusters, the communication cost will increase comparing to that of putting them within the same cluster. The goal is: given the weighted directed graph, how to cluster them into a fixed number of clusters such that a certain objective is minimized.

In [4], Shi and Malik proposed Normalized Cut(*NC*), the algorithm partitions an undirected graph into two disjoint parts so that $F_{ncut} = \frac{W_{1,2}}{W_{1,1}+W_{1,2}} + \frac{W_{2,1}}{W_{2,2}+W_{2,1}}$ is minimized, where $W_{i,j}$ is the sum of the weights of all edges connect the vertices in part i and part j. In addition, when partitioning the graph to $M > 2$ parts, the *NC* minimizes $F_{ncut} = \sum_{i=1}^{M} \frac{\sum_{j\neq i} W_{i,j}}{W_{i,i}+\sum_{j\neq i} W_{i,j}}$. However, what *NC* does is to partition the graph into clusters such that the connections between them are small, it may cause unbalance in the total weight of the edges within each group, especially in the power-law graphs. Since the total weight of the edges within one group can be regarded as the load on one server, *NC* may cause very unbalanced server loads thus is not very suitable for our problem.

In [5], Karypis and Kumar presented a coarsening heuristic named heavy-edge heuristic for which the size of the partition of the coarse graph is within a small factor of the size of the final partition obtained after multilevel refinement. Andreev and Racke in [6] defined a parameter $v > 1$, and considered the problem of partitioning a graph into k components such that no component contains more than $v \cdot \frac{n}{k}$ of the graph and the capacity of the edges between different components of the cut is minimized. The authors presented a bicriteria approximation algorithm which has an approximation ratio of $O(\log^{1.5} n)$ and runs in polynomial time for any constant $n > 1$. Lang[7] tested some popular cluster algorithms on power law graphs, and found that an SDP relaxation avoids the Spectral methods tendency to break off tiny pieces of the graph. In [8], Huang and Nguyen proposed a graph clustering method to quickly partition a large graph into densely connected sub-graphs. They showed the algorithm work well to divide the graph into clusters of the similar sizes. To provide more balanced clusters MinMax Cut [9] was proposed. In [10], Nie et al. proposed to apply additional nonnegative constraint to improve MinMax Cut graph clustering. However, what all of the works considered is to make the size, i.e. the number of nodes in a cluster similar, and it is not the load i.e. the total weights of the edges in a cluster that we seek to balance in our solution, hence they cannot be used in our problem.

In [11], Lui and Chan studied client-server assignment for Client/Server systems, and proposed an efficient partitioning algorithm to make each server process approximately the same amount of workload. The theoretical foundation of their algorithm is based on the linear optimization principle. However, the communication load in [11] only considered the load balance with the assumption that the overall load was same under any client assignment, which differs from our situation. Nishida and Nguyen in [1] studied the client assignment strategy to strike the balance between the total communication load and load balance on servers. They proved that finding the optimal assignment is NP-hard. They devised a heuristic named *BCO-V* to solve the client assignment using relaxed convex optimization. *BCO-V* is a binary splitting method via relaxed convex optimization with considerably expensive time consumption. No theoretical analysis or guarantee was provided either, thus their work was incomplete.

III. Preliminary

Before we introduce our problem formulation and algorithms, we give the brief preliminary of *Semidefinite programming* (SDP), which is the basis of our solution.

First of all we introduce *strict quadratic program*. A quadratic program is the problem of minimizing or maximizing a quadratic function of integer valued variables $(x_1, x_2, ..., x_n)$, subject to quadratic constraints on these variables. If each monomial in the objective function, as well as in every constraint, is of degree 0 (a constraint) or 2, then this is a *strict quadratic program*.

Second, for a *strict quadratic program* in which every integer variable x_i is either $+1$ or -1, we relax each integer variable x_i to a vector variable $\mathbf{v}_i$ such that $\mathbf{v}_i \cdot \mathbf{v}_i = 1$ and $\mathbf{v}_i \in \mathbf{R}^n$, where $\cdot$ means the inner product. After the relaxation the original problem has been transformed to a *vector program*, which is the problem to optimize a linear function of the inner products $\mathbf{v}_i \cdot \mathbf{v}_j, 1 \leq i \leq j \leq n$, subject to linear constraints on these inner products.

Third, when we obtain the *vector program*, we can solve it by SDP [12]. The raw solution obtained by SDP is a matrix, and the $(i,j)^{th}$ entry of the matrix is the inner product $\mathbf{v}_i \cdot \mathbf{v}_j$. Finding the optimal solution to the *vector program* equals to finding the optimal solution of its corresponding SDP, and this SDP can be solved within any error ϵ in time polynomial in n and $log(1/\epsilon)$. Suppose the obtained solution is a matrix $A \in \mathbf{R}^n$, and then by adopting *Cholesky Factorization* on *A*, we obtain the vectors $\mathbf{v}_1, ..., \mathbf{v}_n$, which is the solution to the previous *vector program*.

At last, different rounding method can be used to round each vector $\mathbf{v}_i \in \mathbf{R}^n$ to an integer $x_i \in \{-1, +1\}$ $(i = 1, .., n)$, and $\{x_1, ..., x_n\}$ is the final optimal solution to the original *strict quadratic program*. The rounding methods can be threshold rounding, randomized rounding, etc.

In our paper, we formulate every problem we proposed into a *strict quadratic program*, and then solve it following the procedures above. However some of them cannot be formulated as *strict quadratic program* at first sight, in that case we use some tricks to transform them into equivalent *strict quadratic programs*.

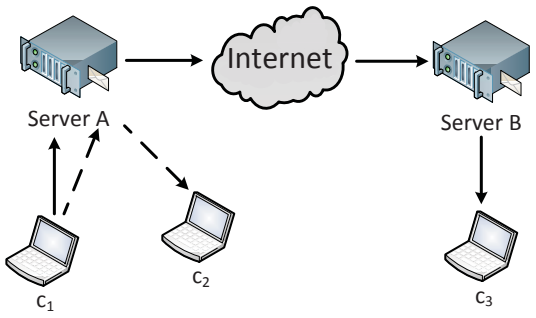


Figure 2. A client/server system with 2 Servers.

IV. Problem Statement with 2-Server

In this section, we study how to assign the clients in the special system with only two servers — Server A (denoted as S_A) and Server B (denoted as S_B). Then we extend the 2-server solution to the case of multiple servers in the next section.

Suppose there are n clients $\{c_1, ...c_n\}$ in the system. For each client c_i, we define a variable x_i to denote which server c_i is assigned to:

$$x_i = \begin{cases} 1, & c_i \text{ is assigned to } S_A \\ -1, & c_i \text{ is assigned to } S_B \end{cases}.$$

We use $r_{i,j}$ to denote the data rate from client i to client j ($r_{i,i} = 0$). As we introduced by Fig. 1, when two clients are assigned to two different servers, the total communication in the system will increase. In detail, for clients c_i and c_j, the communication load between them is $r_{i,j}$ if they are assigned to the same server; when they are assigned to different servers, the communication load doubles, which is $2r_{i,j}$. Since c_i and c_j assigning to the same server means $(x_i, x_j) = (+1, +1)$ or $(-1, -1)$, i.e. $x_i x_j = 1$; while c_i and c_j assigning to different servers means $(x_i, x_j) = (+1, -1)$ or $(-1, +1)$, i.e. $x_i x_j = -1$. The communication load on the servers caused by r_i to r_j is $r_{i,j} + \frac{1-x_i x_j}{2} r_{i,j}$. The total communication is the sum of all the communication loads caused by such pair of clients, hence the total communication load C_t is:

$$C_t = \sum_{i=1}^{n}\sum_{j=1}^{n} r_{i,j} + \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{1 - x_i x_j}{2} r_{i,j}. \quad (1)$$

We denote the approximate load on S_A as C_A and the load on S_B as C_B, based on the definition in Formula 1, we can express the cost on server A as follows:

$$C_A = \sum_{i=1}^{n} \frac{1 + x_i}{4} \sum_{j=1}^{n} r_{i,j} + \sum_{j=1}^{n} \frac{1 + x_j}{4} \sum_{i=1}^{n} r_{i,j} \quad (2)$$

The cost on server B can be expressed as:

$$C_B = \sum_{i=1}^{n} \frac{1 - x_i}{4} \sum_{j=1}^{n} r_{i,j} + \sum_{j=1}^{n} \frac{1 - x_j}{4} \sum_{i}^{n} r_{i,j} \quad (3)$$

Hence, the direct difference between server A and server B can be expressed as $D = |C_A - C_B|$, and:

$$D = |\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{x_i + x_j}{2}r_{i,j}| \tag{4}$$

C_A and C_B are not exactly the loads on server A and server B, respectively. Consider a pair of clients (c_i, c_j) that c_i and c_j are assigned on server A and B respectively, c_i's communication load on A is $r_{i,j}/2$ and c_j's communication load on B is also $r_{i,j}/2$ according to (2) and (3), respectively. But they should both be $r_{i,j}$. In fact, the precise load on each server can be expressed but it is not necessary. The reason is that, although C_A and C_B are not very precise, D is the precise load disparity (the opposite of load balance) between two servers A and B, since all the communication load caused on (c_i, c_j) assigned on two servers separately is offset, and what we need in our problem is D.

To use SDP method, we use $C_b = D^2$ instead of D to denote the load balance. To better explain our objective function, define

$$s_i = \sum_{j=1}^{n}\frac{1}{2}(r_{i,j} + r_{j,i}), \tag{5}$$

then we have:

$$D = |\sum_{i=1}^{n} s_i x_i|, \tag{6}$$

$$\begin{aligned} C_b &= D^2 \\ &= \sum_{i=1}^{n} s_i^2 + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(s_i s_j)(x_i x_j). \end{aligned} \tag{7}$$

Our goal is to strike the balance between the total load C_t and the load balance C_b. We define a parameter $\lambda \in [0, 1]$ to define a weighted combination of C_t and C_b, and then

$$C = \lambda C_t + (1 - \lambda)C_b \tag{8}$$

is the object function. The optimization problem in 2-server case is:

$$\begin{aligned} \min: \quad & C \\ \text{s. t.} \quad & x_i^2 = 1,\ x_i \in \mathbf{Z}. \quad 1 \le i \le n \end{aligned} \tag{9}$$

Define two new parameters:

$$w_{i,j} = -\frac{1}{2}\lambda r_{i,j} + (1 - \lambda)s_i s_j \tag{10}$$

$$\Theta = \frac{3}{2}\lambda\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} r_{i,j} + (1 - \lambda)\sum_{i}^{n} s_i^2 \tag{11}$$

then problem (9) is equivalent to:

$$\begin{aligned} \min: \quad & \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} w_{i,j}x_i x_j + \Theta \\ \text{s. t.} \quad & x_i^2 = 1,\ x_i \in \mathbf{Z}. \quad 1 \le i \le n \end{aligned} \tag{12}$$

It is easy to check that problem (12) is a *strict quadratic problem*, thus it can be solved by the technique we introduced in Section III: Relax each integer variable to transform problem (12) into following vector program:

$$\begin{aligned} \min: \quad & \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} w_{i,j}(\mathbf{v}_i \cdot \mathbf{v}_j) + \Theta \\ \text{s. t.} \quad & \mathbf{v}_i \cdot \mathbf{v}_i = 1,\ \mathbf{v}_i \in \mathbf{R}^n. \quad 1 \le i \le n \end{aligned} \tag{13}$$

Problem (13) can be solved by SDP [12], and the procedure is omitted here. From the above, we have solved the scenario where there are only 2 servers. In the next section, we will solve the general case.

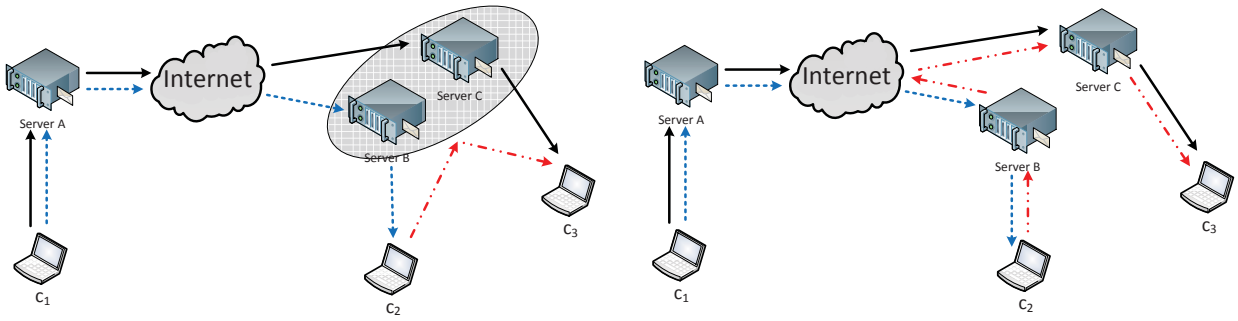
V. Problem Statement with General Case

In general case, we have k servers, $k > 2$. The method in 2 serves may also be adopted to resolve the new problem. The idea is: We first split all the servers into 2 groups, and treat each group as a server, and then apply the 2 server case algorithm to them to allocate every client to one of the server group in a balanced way. After that, for each group, if in it there are more than one servers, we continue splitting this group into two smaller groups and employing the 2 server algorithm again. The recursive treatment continues until every group only contains one server, which means we have assigned every client to a real server, we have completed the task and solve the general case problem.

One of the challenges in recursive algorithm design for the general case is how to split the servers. For the k servers, the most perfect envisage is to split the servers into two groups with equal numbers of servers, since the load is separated equally into two groups and each group has the same amount of servers, we could then assign the clients to the 2 small groups evenly. Afterwards we could split the 2 small groups obtained before to 4 smaller groups with the same size recursively until last. However, this ideal circumstance only happens when k is the power of 2. If k equals to a value other than the powers of 2, we have to seek another method.

To be more specific, in the first splitting, if k is even, then split the servers into two groups each with the equal size of $\frac{k}{2}$ is the current perfect choice. Else if k is odd, we have to split the servers to two groups as evenly as possible. Only in this way the times of recursive server-splitting can be reduced to the lowest. Thus the obvious choice is: one group with size of $\lfloor\frac{k}{2}\rfloor$, and the other with size of $\left(k - \lfloor\frac{k}{2}\rfloor\right)$, which is $\lceil\frac{k}{2}\rceil$. Until every group contains one server the algorithm ends. Fig. 3(a) and Fig. 3(b) show the procedure under 3 servers case. Fig. 3(a) plots the situation after the first splitting, where server B and server C are in a group that is treated as one virtual server. Fig. 3(b) plots the final allocation result, after server B and server C are split.

Another challenge rises along with the recursive procedure. We will show it by Fig. 3(a). After the first splitting, Client c_1 has been separated with c_2 and c_3. In the second splitting, we have to allocate c_2 and c_3 to S_A and S_B. However, even if c_2 and c_3 are allocated to different servers, the communication



(a) The first splitting. (b) The second (final) splitting.

Figure 3. Splitting process in a general case with 3 servers.

cost of link $c_1 \to c_2$ (ends with c_2) or the link $c_1 \to c_3$ (ends with c_3) will not double, since each pair of clients in $c_1 \to c_2$ or $c_1 \to c_3$ has already been separated before the second splitting starts. We call these links appears in subproblems due to recursive splitting as *Extra Link*. When we solve the subproblem of client assignment, with different client assignments, the load balance varies and the *Extra Link* play a role in this variation, total load also varies but *Extra Links* keep the same in total load. Based on this observation, for every client c_i, let e_i denote the sum of the data rates of all the *Extra Links* that start or end with c_i, we modify the formulas (1), (2) and (3) for general case:

$$\widetilde{C}_t = C_t + \sum_{i=1}^{n} e_i \tag{14}$$

$$\widetilde{C}_A = C_A + \sum_{i=1}^{n} \frac{1+x_i}{2} e_i \tag{15}$$

$$\widetilde{C}_B = C_B + \sum_{i=1}^{n} \frac{1-x_i}{2} e_i \tag{16}$$

Suppose we have separated k servers into two groups: A with k_A servers and B with k_B servers ($k_A + k_B = k$), the load difference $\widetilde{D}$ between these two server groups is: $\widetilde{D} = \frac{2}{k}|k_B\widetilde{C}_A - k_A\widetilde{C}_B|$.

It should be noted that $\sum_i s_i = \sum_{i,j}(x_i + x_j)$ (s_i is defined in (5)). We get the following formula:

$$\widetilde{D} = \frac{2}{k}\left|\frac{k}{2}\sum_i (s_i + e_i)x_i - \frac{k_A - k_B}{2}\sum_i (s_i + e_i)\right| \tag{17}$$

Define

$$\widetilde{s}_i = s_i + e_i. \tag{18}$$

If k is even, then we divide k servers equally, which means $k_A = k_B = k/2$. We then substitute s_i in definition (10) and (11) with $\widetilde{s}_i$, obtain the parameters in problem (12), at last use Algorithm 1 to solve it.

If k is odd, in order to minimize $\widetilde{D}$, we have to make $\frac{k}{2}\sum_i (s_i + e_i)x_i$ as close to $\frac{k_A - k_B}{2}\sum_i (s_i + e_i)$ as possible, which means approach $\sum_i \widetilde{s}_i x_i$ to $\frac{k_A - k_B}{k}\sum_i \widetilde{s}_i$. Define

$$\widetilde{C}_b = (\sum_i \widetilde{s}_i x_i)^2 \tag{19}$$

Note that $\widetilde{C}_b$ does not equal to $\widetilde{D}^2$ in general case, however it still plays a significant role as before. To be more specific, the closer $\widetilde{C}_b$ is to $\left(\frac{k_A - k_B}{k}\sum_i \widetilde{s}_i\right)^2$, the closer $\widetilde{D}$ is to 0, and a more balanced client assignment will be found, which is also a better solution for problem (20). Theorem 3 in Theory Analysis section will certify this.

Let

$$O = \left|\frac{k_A - k_B}{k}\right| \sum_{i=1}^{n} \widetilde{s}_i$$

for the sake of concision, the optimization problem under general case can be formulated as:

$$\begin{aligned} \min: \quad & \widetilde{C} = \lambda\widetilde{C}_t + (1-\lambda)|\widetilde{C}_b - O^2| \\ \text{s. t.} \quad & x_i^2 = 1,\ x_i \in \mathbf{Z}.\quad 1 \le i \le n \end{aligned} \tag{20}$$

Recall that $\widetilde{C}_t$ and $\widetilde{C}_b$ are defined in (14) and (19), and $x_i = +1$ or $-1, 1 \le i \le n$. $\forall i, 1 \le i \le n$, we relax x_i in $\widetilde{C}_t$ and $\widetilde{C}_b$ to a vector $\mathbf{v}_i \in \mathbf{R}^n$, and denote the new obtained expressions $\widetilde{C}_t^{\mathbf{v}}$ and $\widetilde{C}_b^{\mathbf{v}}$. The following optimization problem is obtained:

$$\begin{aligned} \min: \quad & \lambda\widetilde{C}_t^{\mathbf{v}} + (1-\lambda)|\widetilde{C}_b^{\mathbf{v}} - O^2| \\ \text{s. t.} \quad & \mathbf{v}_i \cdot \mathbf{v}_i = 1,\ \mathbf{v}_i \in \mathbf{R}^n.\quad 1 \le i \le n \end{aligned} \tag{21}$$

However problem (21) still has an absolute value item in its objective function, which makes it not straightforwardly solvable. We have to transform it. In fact, since $|\widetilde{C}_b^{\mathbf{v}} - O^2|$ equals to $(\widetilde{C}_b^{\mathbf{v}} - O^2)$ if $\widetilde{C}_b^{\mathbf{v}} \ge O^2$, and equals to $(O^2 - \widetilde{C}_b^{\mathbf{v}})$ otherwise, we can solve the following two problems:

$$\begin{aligned} \min: \quad & \lambda\widetilde{C}_t^{\mathbf{v}} + (1-\lambda)(\widetilde{C}_b^{\mathbf{v}} - O^2) \\ \text{s. t.} \quad & \mathbf{v}_i \cdot \mathbf{v}_i = 1,\ \mathbf{v}_i \in \mathbf{R}^n.\quad 1 \le i \le n \\ & \widetilde{C}_b^{\mathbf{v}} \ge O^2 \end{aligned} \tag{22}$$

$$\begin{aligned} \min: \quad & \lambda\widetilde{C}_t^{\mathbf{v}} + (1-\lambda)(O^2 - \widetilde{C}_b^{\mathbf{v}}) \\ \text{s. t.} \quad & \mathbf{v}_i \cdot \mathbf{v}_i = 1,\ \mathbf{v}_i \in \mathbf{R}^n.\quad 1 \le i \le n \\ & \widetilde{C}_b^{\mathbf{v}} \le O^2 \end{aligned} \tag{23}$$

After solving the two sub-problems: *vector programs* (22) and (23) using SDP, we compare the two object function values of them, and then the one with the smaller value is chosen, and the corresponding $(\mathbf{v}_1, ...\mathbf{v}_n)$ is the final solution to the relaxed problem (21). After that we use the rounding method to obtain the final solution $(x_1, ..., x_n)$ for (20).

VI. Algorithm

Based on the details we presented in section (IV) and (V), we present the algorithms for the 2 server case and the general case. Algorithm 1 is for the case where there only exist 2 servers and Algorithm 2 is used to deal with the general circumstances where there are more than 2 servers. We use *BSP* to uniformly denote our algorithms since they are Binary splitting methods based on solving SDP.

Algorithm 1 *BSP* for 2-Server Case

Step 1. Compute the value of $w_{i,j}$ and C.
Step 2. Solve *vector program* (13), obtain the solution $(\mathbf{v}_1, ..., \mathbf{v}_n)$.
Step 3. Randomly pick a vector $\mathbf{r}$, which to be a uniformly distributed vector in the n-dimensional unit sphere.
Step 4. for $1 \leq i \leq n$
if$(\mathbf{v}_i \cdot \mathbf{r} \geq 0)$ let x_i = 1, else let $x_i = -1$

Algorithm 2 *BSP* for General Case

Step 1. if k is even, divide the servers into two groups with equal size as $\frac{k}{2}$, else go to Step 3.
Step 2. use Algorithm 1 with $\widetilde{s}_i$ instead of s_i. If the each server group contains more than one servers, return to step 1 to recursively split the two server subgroups.
Step 3. divide k servers into two groups, one with size $\lceil \frac{k}{2} \rceil$, and the other with $\lfloor \frac{k}{2} \rfloor$.
Step 4. Solve *vector programs* (22) and (23) by SDP. Let $(\mathbf{v}_1, ..., \mathbf{v}_n)$ be the optimal solution, which belongs to one of the two *vector programs* that has the smaller objective value.
Step 5. Use the method in step 4 in Algorithm 1 to obtain the client allocation result $(x_1, ..., x_n)$ on the two virtual server groups from optimal solution $(\mathbf{v}_1, ..., \mathbf{v}_n)$ obtained in step 4.
Step 6. Repeat step 1 to step 6 for each of the two server subgroups, until each group contains only one server.

A. Time Complexity

Our proposed algorithm is based on SDP, and a SDP with n variables (clients) can be solved within any error ϵ in time polynomial in n and $\log(1/\epsilon)$. We fix ϵ and then denote the time SDP takes for solving a problem with n clients by $f(n)$. The time complexities for 2-server case and general k-server case are $O(n + f(n))$ and $O((n + f(n)) \log k)$ respectively. Note that SDP is a subfield of convex optimization, for a fix ϵ, the time complexities of SDP and convex optimization are the same. *BCO-V* is based on convex optimization, and its time complexity is $O(n(n + f(n)) \log k)$, which is more than an order of magnitude greater than *BSP*'s running time.

VII. Theory Analysis

In this section we give the approximation ratio and some bounds of our algorithms.

Lemma 1.

$$\mathbf{Pr}[\mathbf{v}_i \textit{ and } \mathbf{v}_j \textit{ are separated}] = \frac{\theta_{ij}}{\pi}$$

where θ_{ij} is the angle from $\mathbf{v}_i$ to $\mathbf{v}_j$, in other words, $\theta_{ij} = arccos(\mathbf{v}_i \cdot \mathbf{v}_j)$.

Let $\alpha = 0.87$, we have Lemma 2 and Lemma 3.

Lemma 2.

$$\frac{\theta}{\pi} \geq \alpha \left(\frac{1 - cos\theta}{2} \right)$$

Lemma 3.

$$1 - \frac{\theta}{\pi} \geq \alpha \left(\frac{1 + cos\theta}{2} \right)$$

The proofs for Lemma 1, Lemma 2 and Lemma 3 can be found in [13].

Lemma 4. *Define $S = \sum_{i=1}^{n} s_i^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} s_i s_j$, let $\mathbf{E}[\mathbf{B}]$ be the expectation of the value of C_b obtained by Algorithm 1, then*

$$\mathbf{E}[\mathbf{B}] - S \leq \alpha(C_b^{\mathbf{v}} - S)$$

Proof:

$$\begin{aligned}
\mathbf{E}[\mathbf{B}] - S &= \mathbf{E}[C_b - (\sum_{i=1}^{n} s_i^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} s_i s_j)] \\
&= \mathbf{E}[\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{x_i x_j - 1}{2} 2 s_i s_j] \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} -\mathbf{Pr}[\mathbf{v}_i \text{ and } \mathbf{v}_j \text{ are separated}] 2 s_i s_j \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} -\frac{\theta_{ij}}{\pi} 2 s_i s_j
\end{aligned}$$

and

$$C_b^{\mathbf{v}} - S = \sum_{i=1}^{n} \sum_{j=1}^{n} -\frac{1 - \mathbf{v}_i \mathbf{v}_j}{2} 2 s_i s_j$$

Because $cos\theta_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$, according to Lemma 2, $-\frac{\theta_{ij}}{\pi} \leq \alpha(-\frac{1 - \mathbf{v}_i \mathbf{v}_j}{2})$, and since $2 s_i s_j \geq 0$, Lemma 4 holds. ■

Lemma 5. *Define $R = 2 \sum_{i=1}^{n} \sum_{j=1}^{n} r_i r_j$, let $\mathbf{E}[\mathbf{T}]$ be the expectation of the value of C_t obtained by Algorithm 1, then*

$$\mathbf{E}[\mathbf{T}] - R \leq \alpha(C_t^{\mathbf{v}} - R)$$

Proof:

$$\begin{aligned}
\mathbf{E}[\mathbf{T}] - R &= \mathbf{E}[C_t - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} r_i r_j] \\
&= \mathbf{E}[\sum_{i=1}^{n} \sum_{j=1}^{n} -\frac{(1 + x_i x_j)}{2} r_i r_j] \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} -\mathbf{Pr}[x_i \text{ and } x_j \text{ are not separated}] r_i r_j \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} -(1 - \frac{\theta_{ij}}{\pi}) r_i r_j
\end{aligned}$$

and

$$C_t^{\mathbf{v}} - R = \sum_{i=1}^{n} \sum_{j=1}^{n} -\left(\frac{1 + \mathbf{v}_i \mathbf{v}_j}{2} \right) r_i r_j$$

Because $cos\theta_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$, according to Lemma 3, $-(1 - \frac{\theta_{ij}}{\pi}) \leq \alpha(-\frac{1+\mathbf{v}_i\mathbf{v}_j}{2})$, and since $r_i r_j \geq 0$, Lemma 5 holds. ■

Theorem 1. *Let* $\mathbf{E}[\mathbf{W}]$ *be the expectation of the objective value of (12) obtained by algorithm 1,* $\mathbf{OPT}_v$ *be the optimal objective value of (13), and* $W_D = \lambda R + (1-\lambda)S$*, then*

$$\mathbf{E}[\mathbf{W}] - W_D \leq \alpha(\mathbf{OPT}_v - W_D)$$

Proof:

$$\begin{aligned} \mathbf{E}[\mathbf{W}] - W_D &= \lambda(\mathbf{E}[\mathbf{T}] - R) + (1-\lambda)(\mathbf{E}[\mathbf{B}] - S) \\ &\leq \lambda\alpha(C_t^{\mathbf{v}} - R) \\ &\quad + (1-\lambda)\alpha(C_b^{\mathbf{v}} - S) \\ &= \alpha(\mathbf{OPT}_v - W_D) \end{aligned}$$

Lemma 4 and 5 can help to get "≤". ■

From Theorem 1, we obtain the approximation ratio of Alg. 1, which is $\alpha = 0.87$. Note that the approximation ratio is the one in a broader sense since W_D has to be used to establish the inequation. However it provides a bound for our algorithm's performance, which makes Alg. 1 the first bounded approximation algorithm for similar problems.

Lemma 6. *Define* $\widetilde{S} = \sum_{i=1}^{n} \widetilde{s}_i^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} \widetilde{s}_i\widetilde{s}_j$*, let* $\mathbf{E}[\widetilde{\mathbf{B}}]$ *be the expectation of the value of* $\widetilde{C}_b$ *obtained by Algorithm 2, then*

$$\mathbf{E}[\widetilde{\mathbf{B}}] - \widetilde{S} \leq \alpha(\widetilde{C}_b^{\mathbf{v}} - \widetilde{S})$$

Proof: Similar to the proof of Lemma 4. ■

Lemma 7. *Define* $\widetilde{S}' = \frac{1+\alpha}{1-\alpha}O^2 + \widetilde{S}$*, and let* $\mathbf{E}[\widetilde{\mathbf{B}}']$ *be the expectation of the value of* $\left|\widetilde{C}_b - O^2\right|$ *obtained by Algorithm 2, then*

$$\mathbf{E}[\widetilde{\mathbf{B}}'] - \widetilde{S}' \leq \alpha\left(|\widetilde{C}_b^{\mathbf{v}} - O^2| - \widetilde{S}'\right)$$

Proof:

$$\begin{aligned} \mathbf{E}[\widetilde{\mathbf{B}}'] - \widetilde{S}' &\leq \mathbf{E}[\widetilde{\mathbf{B}}] + O^2 - \widetilde{S}' \\ &\leq \alpha(\widetilde{C}_b^{\mathbf{v}} - \widetilde{S}) + \widetilde{S} + O^2 - \widetilde{S}' \\ &\leq \alpha\left(|\widetilde{C}_b^{\mathbf{v}} - O^2| + O^2 - \widetilde{S}\right) \\ &\quad - \frac{\alpha}{1-\alpha}O^2 \\ &= \alpha\left(|\widetilde{C}_b^{\mathbf{v}} - O^2| - \widetilde{S}'\right) \end{aligned}$$

We can get the first "≤" by Lemma 6. ■

Lemma 8. *Define* $\widetilde{R} = 2\sum_{i=1}^{n}\sum_{j=1}^{n} r_i r_j + \sum_{i=1}^{n} e_i$*, let* $\mathbf{E}[\widetilde{\mathbf{T}}]$ *be the expectation of the value of* $\widetilde{C}_t$ *obtained by Algorithm 2, then*

$$\mathbf{E}[\widetilde{\mathbf{T}}] - \widetilde{R} \leq \alpha(\widetilde{C}_t^{\mathbf{v}} - \widetilde{R})$$

Proof: Similar to the proof of Lemma 5. ■

Theorem 2. *Let* $\mathbf{E}[\widetilde{\mathbf{W}}]$ *be the expectation of the objective value of (12) obtained by Algorithm 2,* $\widetilde{\mathbf{OPT}}_v$ *be the optimal objective value obtained by solving (22) and (23), and* $\widetilde{W}_D = \lambda\widetilde{R} + (1-\lambda)\widetilde{S}'$*, then*

$$\mathbf{E}[\widetilde{\mathbf{W}}] - \widetilde{W}_D \leq \alpha(\widetilde{\mathbf{OPT}}_v - \widetilde{W}_D)$$

Proof: Use Lemma 7 and Lemma 8, similar to the proof of Theorem 1. ■

From Theorem 2, we obtain the bound of Alg. 2 in each iteration, with the approximation ratio $\alpha = 0.87$ in a broader sense. Because Alg. 2 needs $O(\log k)$ iterations, Alg. 2 is bounded with a factor $0.87^{\log k}$ in a broader sense.

Theorem 3. *In general case, suppose* $k_A \geq k_B$*, if there exists an optimal solution* $(x_1, ...x_n)$ *satisfies* $|\widetilde{C}_b - O^2| = \epsilon$*, then there must exist an optimal solution* $(x'_1, ...x'_n)$ *satisfies* $\widetilde{D}' \leq O^{-1}\epsilon$*.*

Proof: If $\sum_{i=1}^{n} \widetilde{s}_i x_i \geq 0$, let $x'_i = x_i,\ i = 1, 2, ..., n$. Since $\widetilde{C}_b = \left(\sum_{i=1}^{n} \widetilde{s}_i x_i\right)^2$, $|\sum_i \widetilde{s}_i x'_i - O| = \frac{\epsilon}{|\sum_{i=1}^{n} \widetilde{s}_i x'_i + O|} \leq O^{-1}\epsilon$. Thus we have $\widetilde{D}' = \frac{2}{k} \cdot \frac{k}{2} |\sum_{i=1}^{n} \widetilde{s}_i x'_i - O| \leq O^{-1}\epsilon$.

Else let $x'_i = -x_i,\ i = 1, 2, ..., n$. After this step the objective value of problem (20) keeps the same, which makes $(x'_1, ..., x'_n)$ still an optimal solution with $\sum_{i=1}^{n} \widetilde{s}_i x'_i \geq 0$. From the previous case it is true. Theorem 3 holds. ■

What Theorem 3 proves is the validity of solving problem (20). Recall that $\widetilde{D}'$ defined by (17) is the load balance in general case. Theorem 3 tells us that minimizing $|\widetilde{C}_b - O^2|$ in (20) is equivalent to minimizing $\widetilde{D}'$, hence solving optimization problem (20) can seek the client assignment with balanced server loads in general case.

VIII. Simulation

To test the performance of our algorithms, we compared the effect of other popular algorithms with our *BSP*. The other three algorithms we use to compare with are:

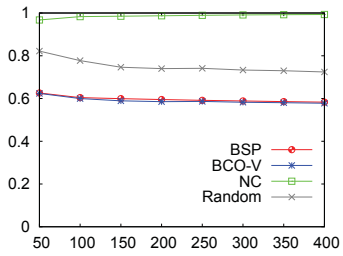
BCO-V: The binary splitting via relaxed convex optimization [1].

NC: Normalized Cut [9].

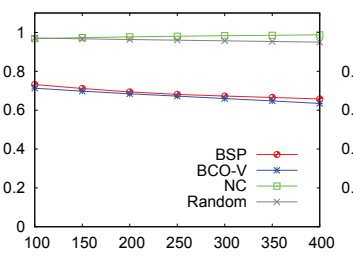
Random: The clients are randomly allocated to each server.

A. Experiment Setup

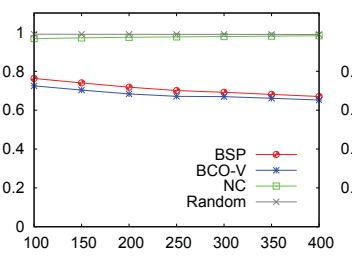
For *BSP* and *BCO-V*, MATLAB is our simulation platform and we implement both of their code. The tool we use in our MATLAB code is CVX, a package for specifying and solving convex programs [14],[15]. For the *NC* algorithm, we use MATLAB code [16]. In our simulation, we set $\lambda = 0.5$ for all formulas. For the parameter β in *BCO-V*, we also set it as 0.5. To obtain the message rate $r_{i,j}$ between the clients, we randomly generate hundreds of graphs by our random graph generator, in every graph each vertex is allocated at most 20 randomly selected neighbors, and randomly assign weight to each edge (i, j), which is $r_{i,j}$. For two nodes i and j that are not adjacent, $r_{i,j} = 0$. Before every experiment starts, $r_{i,j}$ is normalized to make $\sum_{i=1}^{n}\sum_{j=1}^{n} r_{i,j} = 1$.



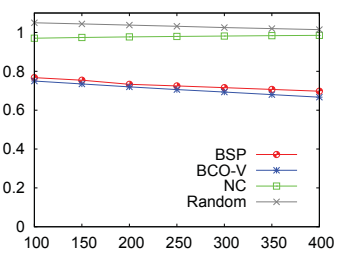
(a) C



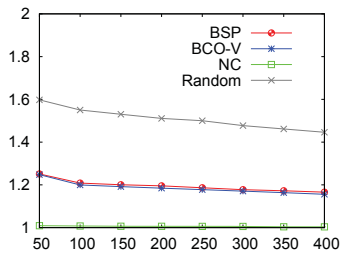
(a) $\widetilde{C}$



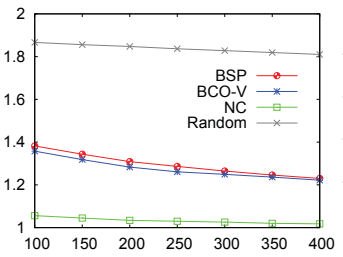
(a) $\widetilde{C}$



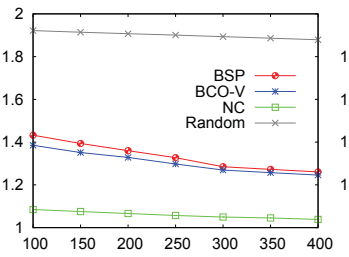
(a) $\widetilde{C}$



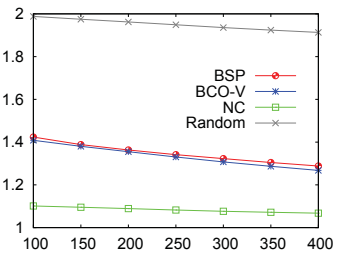
(b) C_t



(b) $\widetilde{C}_t$



(b) $\widetilde{C}_t$



(b) $\widetilde{C}_t$

Figure 4. $k = 2$ servers.

Figure 5. $k = 5$ servers.

Figure 6. $k = 7$ servers.

Figure 7. $k = 10$ servers.

For each sub-figure in Fig. 4-7: the abscissa is the number of clients, the ordinate is the value indicated by its caption.

B. Experiment for 2 Server Case

In this subsection, we present the result of our experiment of Algorithm 1. Here, $C_b = D^2$. The results of C obtained by different algorithms is shown in Fig. 4(a), and the total load C_t is shown in Fig. 4(b). Since performances of different algorithms vary too much on balancing load, we use Table I to present C_b. Noted that for C, C_t and C_b, the smaller value means the algorithm works better.

Table I. Load balance D^2 with $k = 2$ servers

#clients	*BSP*	*BCO-V*	*NC*	Random
100	0.0156^2	0.0051^2	0.9796^2	0.0681^2
200	0.0119^2	0.0037^2	0.9801^2	0.0632^2
300	0.0097^2	0.0013^2	0.9832^2	0.0555^2
400	0.0073^2	0.0008^2	0.9912^2	0.0497^2

Before the analysis of the simulation result, we have to emphasize that due to our pretreatment, $C_t \in [1, 2]$, and $C_b \in [0, 1]$. Our algorithm works very well in this case. From the result we can see that regardless of the number of clients, our algorithm *BSP* constantly gives nearly optimal solutions. *BCO-V* is the most optimal solution so far, it works always the best yet with enormous time consumption, which will be compared afterwards. *BSP* works fairly close to *BCO-V*, all the value of F, F_t and F_b obtained by *BSP* is always less than 101% of those obtained by *BCO-V*. *NC* gives very good C_t, however it loses greatly in balancing load, thus gives very bad C_b and C. From our simulation, we also found that *NC* algorithm is prone to isolate the nodes that have weak connection to others. Random method computes acceptable load balance C_b, but works worst in reducing the total communication cost C_t.

C. Experiment for General Case

We test Algorithm 2 by setting the number of servers to $5, 7, 10$, and increase the number of clients from 100 to 400. The results of objective value $\widetilde{C}$ and total load $\widetilde{C}_t$ are plotted in Fig. 5-7. In general case, in order to compare our algorithm with *BCO-V*, we use Gini coefficient to evaluate the load balance on multiple servers:

$$G_b = \frac{1}{k-1}\left(\frac{2\sum_{a=1}^{k} a l_a}{\sum_{a=1}^{k} l_a} - k - 1\right),$$

where $l_1 \le l_2 \le \cdots \le l_k$. $G_b \in [0, 1]$. The experimental result of G_b is plotted in Table II. *BSP* lowers both $\widetilde{C}_t$ and

Table II. Load balance G_b

$k = 5$				
#clients	*BSP*	*BCO-V*	*NC*	Random
100	0.0819	0.0682	0.8805	0.1014
200	0.0787	0.0852	0.9207	0.1223
300	0.0809	0.0601	0.9403	0.1066
400	0.0853	0.0472	0.9571	0.1148
$k = 7$				
#clients	*BSP*	*BCO-V*	*NC*	Random
100	0.0938	0.0649	0.8519	0.0992
200	0.0832	0.0411	0.8839	0.0935
300	0.0955	0.0701	0.9083	0.1076
400	0.0805	0.0581	0.9275	0.1003
$k = 10$				
#clients	*BSP*	*BCO-V*	*NC*	Random
100	0.1092	0.0916	0.8393	0.1115
200	0.1029	0.0849	0.8679	0.1132
300	0.1091	0.0790	0.8864	0.1143
400	0.1119	0.0663	0.9023	0.1257

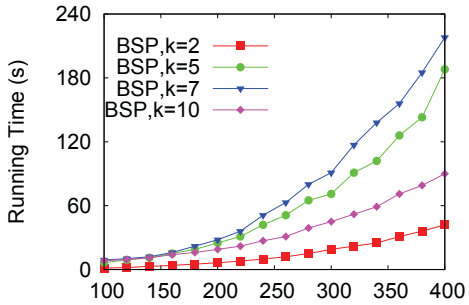


Figure 9. Running Time of *BSP*.

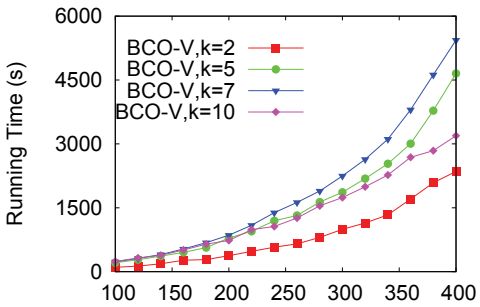


Figure 10. Running Time of *BCO-V*.

G_b, it gives very low total communication cost and fairly balanced load. *BCO-V* still gives excellent result on $\widetilde{C}_t$ and G_b. *NC* yields low $\widetilde{C}_t$, but it works badly on load balancing, i.e. produces very high G_b. Random, on the other hand, yields a fairly balanced load distribution, however it fails to reduce the total balance. In all, our algorithm *BSP* obtains low objective value $\widetilde{C}$, which is approximately 105% of the best yet time-cost algorithm *BCO-V*.

D. Running Time Comparison

From the above result we see that our algorithm *BSP* performances well on reducing the total communication cost, while keeps the variation of loads on different servers in a relatively low level. As we mentioned before, *BCO-V* is an optimal solution to solve the client-server assignment problem, and *BSP* works highly close to the best known algorithm, *BCO-V*. The next criterion we study is the running time, which directly impacts the feasibility of an algorithm.

Through simulation, we found that *BSP* saves time comparing to *BCO-V*. We see that when $k = 2$, *BSP*'s runtime is approximately $\frac{1}{60}$ of *BCO-V*'s regardless of the scale of the clients. When $k = 5$, $k = 7$ and $k = 10$, *BSP*'s runtime is $\frac{1}{25}$, $\frac{1}{27}$ and $\frac{1}{35}$ of *BCO-V*'s, respectively. In the simulation we also found an interesting fact that the increasing of servers' quantity does not necessarily increase the time consumption. When $k = 10$ and $n = 400$, both of the runtime of *BSP* and *BCO-V* is less than that in the case where $k = 5$ and $n = 400$. The reason is that when $k = 10$ and $n = 400$, the original problem is quickly divided into two subproblems with 5 servers, each of the subproblem has approximately 200 clients. Hence the process next is to solve the two small case where $k = 5$ and $n = 200$, which is shorter than the time it takes to solve the case where $k = 5$ and $n = 400$.

IX. Conclusion

In this paper, we discuss the client assignment problem in client/server systems, which is one kind of internet distributed systems. This problem has been proved to be NP-hard, and we present our new mathematical mode for it. Base on our model, we propose the SDP based approximation algorithm *BSP*, and prove that each iteration of *BSP* is bounded by a factor 0.87 in a broad sense. We are the first to give the algorithm with theoretical analysis for optimization problem of striking the balance between the total load and load balance in client/server systems, and its time complexity is also theoretically an order of magnitude smaller than that of *BCO-V*, the best known algorithm. During the simulation, we confirm that *BSP* works almost as well as *BCO-V* on reducing the total load and balancing the loads of different servers, but evidently outperforms it in terms of running time.

References

[1] H. Nishida and T. Nguyen. Optimal client-server assignment for internet distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1–1, 2012.

[2] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[3] The xmpp standards foundation, xmpp, http://xmpp.org/about/.

[4] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[5] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[6] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 120–124, New York, NY, USA, 2004. ACM.

[7] Kevin Lang. Finding good nearly balanced cuts in power law graphs. Technical report, 2004.

[8] Mao Lin Huang and Quang Vinh Nguyen. A fast algorithm for balanced graph clustering. In *Information Visualization, 2007. IV '07. 11th International Conference*, pages 46–52, 2007.

[9] M. Schlag P. Chan and J. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(PrePrints), Sep. 2012.

[10] Feiping Nie, Chris Ding, Dijun Luo, and Heng Huang. Improved min-max cut graph clustering with nonnegative relaxation. In *Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part II*, ECML PKDD'10, pages 451–466, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] C. S. Lui and M. F. Chan. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transction on Parallel Distributed System*, 13(3):193–211, March 2002.

[12] D.-Z. Du, K.-I. Ko, and X. Hu. Design and analysis of approximation algorithms. 2012.

[13] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42(6):1115–1145, 1995.

[14] Inc. CVX Research. CVX: Matlab software for disciplined convex programming, version 2.0 beta, http://cvxr.com/cvx, September 2012.

[15] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[16] Normalized Cuts Segmentation Code for MATLAB. http://www.timotheecour.com/software/ncut/ncut.html.