Received 10 October 2013; revised 28 February 2014 and 11 July 2014; accepted 7 September 2014.
Date of publication 23 September 2014; date of current version 4 February 2015.

Digital Object Identifier 10.1109/TETC.2014.2358801

Distributed Client-Server Assignment for Online Social Network Applications

THUAN DUONG-BA[1], (Student Member, IEEE), THINH NGUYEN[1], (Member, IEEE), BELLA BOSE[1], (Fellow, IEEE), AND DUC A. TRAN[2], (Member, IEEE)

[1]Department of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA
[2]Department of Computer Science, University of Massachusetts at Boston, Boston, MA 02125 USA
CORRESPONDING AUTHOR: T. DUONG-BA (duongba@eecs.oregonstate.edu)

ABSTRACT We study the problem of assigning users to servers with an emphasis on the distributed algorithmic solutions. Typical online social network applications, such as Facebook and Twitter, are built on top of an infrastructure of servers that provides the services on behalf of the users. For a given communication pattern among users, the loads of the servers depend critically on how the users are assigned to the servers. A good assignment will reduce the overall load of the system while balancing the loads among the servers. Unfortunately, this optimal assignment problem is NP-hard. Therefore, we investigate three heuristic algorithms for solving the user server assignment problem: 1) the centralized simulated annealing (CSA) algorithm; 2) the distributed simulated annealing (DSA) algorithm; and 3) the distributed perturbed greedy search (DPGS). The CSA algorithm produces good solution in the fastest time, however it relies on timely accurate global system information, and is practical only for small and static systems. In contrast, the two distributed algorithms, DSA and DPGS, exploit local information at each server during their search for the optimal assignment, and thus can scale well with the number of users and servers as well as adapting to the system dynamics. Simulation results show that the performance of the distributed algorithms, specifically the DPGS algorithm, is very competitive with that of the centralized algorithm while providing the advantage of naturally adapting to time-varying communication patterns of users.

INDEX TERMS Distributed algorithms, online social network, operations research, combinatorial optimization.

I. INTRODUCTION

Social network applications such as Facebook, Twitter, and many popular Internet applications [1] employ an underlying infrastructure of servers that facilitate communications among the users. In these settings, the users communicate with each other indirectly via their designated servers. Specifically, for an online social network system (OSN) such as Facebook, each user profile and its data are stored at its primary servers. When a user u posts a message, the message is first sent to its designated server S_u. Suppose a user v is a friend of user u, then whenever v logs in his or her (Facebook) account, the updated message will be pushed directly from server S_u to v if v happens to be assigned to the same server S_u. Otherwise if user v was previously assigned to a different server S_v, then a read request is sent from server S_v to server S_u; server S_v then receives the updated message from S_u. Finally, user v reads the updated message from server S_v. For private and instant messages embedded in OSNs, the mechanism is slightly different. User u first sends a message to server S_u, server S_u then forwards the message to user v if he or she is located on the same server S_u; otherwise, server S_u then forwards the message to server S_v which then forwards the message to user v. Effectively, the two designated servers S_u and S_v communicate with each other on behalf of their users. This indirect communication architecture enables rich application functionalities that is often difficult otherwise. Last but not least, this architecture also scales well with the number of users. Specifically, new servers can be added incrementally to accommodate the new users. That said, this indirect communication architecture requires additional resources for handling the inter-communication among servers. In what follows, we will elaborate more on the user server assignment problem.

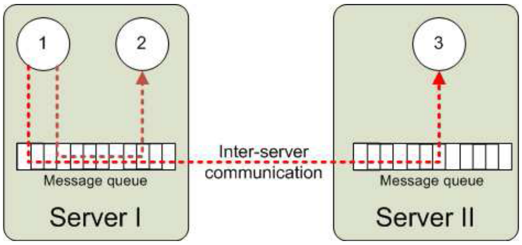


FIGURE 1. Message exchange mechanism. (a) User communication Example. (b) Balanced partitions. (c) Improved scheme. (d) Best resource utilization.

The indirect communication mechanism is depicted in Figure 1 where each server has a message queue for processing user messages. If two users are located on the same server, e.g., user 1 and user 2 are located on server I, then each message is processed once by the server I's. If the two users are located on two different servers, e.g., user 1 and user 3, then each message will have to be processed by both servers I and II. In addition, there is an inter-server communication load between the two servers.

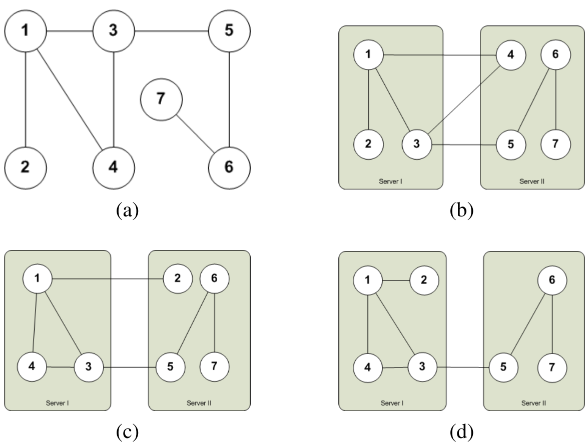


FIGURE 2. Examples of assignment schemes.

Let's consider an example of user communication pattern as shown in Figure 2(a). Nodes represent users and an edge between two nodes represents the communication between the two users. The weight of the edge represents how often the users communicate with each other. In this example, we assume that all users communicate at equal rate, so all the edge weights are normalized to 1. The assignment scheme shown in Figure 2(b) has a perfectly balanced load where each server processes the same number of messages, i.e., five messages per unit time. The total load incurred to all the servers will be 10 messages per unit time. Due to the inter-server communication, of these 10 messages, there are three messages per unit time processed by both servers.

Now, consider a less balanced assignment in shown Figure 2(c). The total load is now 9 messages per unit time, of which, only two messages per unit time are due to inter-communication. We can further reduce the inter-server communication by moving user 2 to server I. The resulted configuration is shown Figure 2(d) which has imbalanced load at servers but smaller total load.

In many large scale systems, the problem of client-server assignment with a specific objective of reducing the total load while maintaining a good load balance among the servers is crucial. We assume the load at the servers is due mainly to handling messages/information passing among users. We begin with the following observations:

1) The amount of load incurring by messages exchanged between two users assigned to the same server will be less than that of when the users are assigned to different servers. This is due to local communication incurred less load than non-local communications. This observation will be made more precisely in Section III. Consequently, the total load is minimum when all users are assigned to one server.
2) However, when all users are assigned to one server, this server might be overloaded, resulting in degraded performance. Also, from the robustness perspective, this assignment is prone to bottleneck failures.
3) It is beneficial to assign two users that exchange messages with each other often to the same server in order to minimize the overall communication load. On the other hand, because of the load balance consideration, two users that rarely exchange messages with each other should be assigned to two different servers.

Based on the observations above, the communication pattern among the users is critical to the optimal client-server assignment. We must strike a balance between reducing the overall communication load, equivalently the communication cost among the servers, and increasing load fairness among the servers, i.e., the load balance. From a broad perspective, the client/server assignment problem can be viewed as a special class of graph partitioning problems that are harder to solve. That said, there is a vast literature on graph partitioning problems [2]–[8] originated from various applications including job scheduling and image segmentation. These methods will be discussed and contrasted with our algorithms in more detail in the Section II. We note that many of these existing techniques require centralized processing and are known to be very computationally expensive and produce local optimal solutions only. The contributions of this paper include three heuristic algorithms for solving the user/server assignment problem: 1) The Centralized Simulated Annealing (CSA) algorithm, 2) the Distributed Simulated Annealing (DSA) algorithm, and 3) the Distributed Perturbed Greedy Search (DPGS). The CSA algorithm produces good solution in the fastest time, however it relies on timely accurate global system information, and is practical only for small and static systems. In contrast, the two distributed algorithms, DSA and DPGS, exploit local information at each server during their search for the optimal assignment. Specifically, in the DSA and DPGS algorithms, the servers exchange and update information about their loads iteratively among themselves. Based on these, the servers make the decisions to reassign their users to others in such a way that the final client-server assignment is approximately optimal. Simulation results show that the performance of the distributed algorithms, specifically DPGS algorithm, is very competitive with that of the centralized algorithm while providing the advantage of naturally adapting to time-varying communication patterns of users.

Our paper is organized as follows. In Section II, we briefly discuss some related work. In Section III, we present a mathematical model and optimization formulation for the client-server assignment problem. Key ingredients used for the proposed distributed algorithm will be derived. Next, we describe the CSA algorithm in Section IV-A, the DSA algorithm in Section IV-C and the DPGS algorithm in Section IV-D. In Section V, we present the simulation results of the proposed algorithms for different patterns of user communication. Finally, we provide a few concluding remarks in Section VI.

II. RELATED WORK

The client-server assignment problem can be viewed as an instance of the k-way graph partitioning problem. The k-way graph partitioning problem is defined as follows: Given a weighted graph $G = (V, E)$ with $w(e)$ and $w(v)$ denoting the weight of edge $e \in E$ and the weight of vertex $v \in V$, respectively. The problem is to partition V into k subsets, $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \emptyset \, \forall i \neq j$ and $w(V_i) \approx \frac{w(V)}{k}$.

The general k-way graph partitioning problem has been shown to be NP-hard [9]. In other words, there is currently no known polynomial time algorithm to obtain the exact optimal solution, and it is likely to remain so. Therefore, a large part of the literature on the k-way partitioning problem have been focused on developing heuristic algorithms and determining special conditions for which exact solutions can be obtained. Notably, the k-way partitioning problem have been widely studied in very-large-scale integration (VLSI) applications, and many heuristic algorithms have been developed. Popular algorithms include the Kerninghan-Lin algorithm (KL) [10] and the Fiduccia-Mattheyes algorithm (FM) [2]. Almost all of these heuristic algorithms can be classified into a number of approaches, ranging from simulated annealing (SA) [11]–[13] and genetic algorithms [14], to large-step Markov chain [15] and spectral methods [3]–[5], to community structure-based methods [16] and graph combinatoric and multi level schemes [3], [17]. It is important to note that while all of these algorithms attempt to solve the general underlying k-way partition problem, they often exploit the structures specific to certain applications in order to improve solution quality or time complexity. In [6], authors emphasized on minimizing the largest inter-group flow in order to solve the clustering problem. In [5], [18], [19]–[20], the proposed algorithms focus on balancing the quantity of elements in each cluster without considering the effects of edges cut on weights of elements. Applications of k-way graph partitioning problem in balancing distributed systems have been proposed in previous works that address the tasks scheduling/allocation problems [7], [8], [21]. In tasks scheduling and/or allocation problems, however, the total load incurred by execution tasks at processors are independent with the inter-communication load [22]–[24]. In general, existing graph partitioning and task schedule algorithms focus on the problem that attempts to balance the size of each partition while minimizing the weights of the associated edges across the cuts. For example, Gracelus balances the partition size, i.e., number of users, but not the total weights of the associated edges within a partition. The classic min-cut algorithm finds the cut that results in minimum cut weights but ignoring the weights within a partition. These algorithms are not appropriate to model the OSN applications. Our work focuses on the problems in which the load distributed at servers is effected by the inter-server communication, especially OSN-like systems. Additionally, our proposed algorithms are based on iterative methods and can be executed in a distributed manner.

Current OSN and distributed database systems, i.e., Cassandra [1], use random paritioner schemes [25] to organize user data in the cluster node ring. The random paritioner is implemented based on the consistent hashing algorithm [26] that chooses how data is stored on a particular node. The random partitioner focuses on the ease of addressing user data, assigning a roughly equal number of users to each server and ignoring the social relationships among the users. Most relevant to our work is that of Nishida and Nguyen [27]. In this work, the authors exploit the specific structures of the client-server problem and reformulate it as a relaxed convex optimization problem. They then propose a *centralized* hierarchical relaxed convex optimization algorithm for obtaining an approximate solution. Unlike ours, their work is not focused on distributed algorithms as well as the network dynamics. An extension of client-server assignment problem is the replication problem proposed in [28] and [29]. In this setting, each user is assigned to more than one server in order to reduce the read and write costs. Importantly, most of these algorithms are classified as centralized algorithms since they assume the global knowledge. While centralized algorithms are appropriate for many applications, in general, they are not suitable in distributed dynamic settings involving a large number of users.

III. PROBLEM FORMULATION

Our objective is to find an approximately optimal client-server assignment that results in small total communication load while maintaining a certain level of load balance. This objective will be made precise shortly with a mathematical optimization formulation. Firstly, we summarize the notations to be used throughout the paper:

- M: Number of users
- N: Number of servers
- $G = (V, E)$: An edge-weighted graph represents the communication pattern among the users, where V denotes the set of all users, $|V| = M$. Each vertex represents a user. E denotes the set of all communication among the users. An edge between two vertices v_i and v_j represents a communication between two users u and v.

- $A^{M \times M}$: An adjacent matrix induced by the graph G. $A_{uv} = 0$ if users u and v never communicate. $A_{uu} = 0$ since u does not communicate with itself. $A_{uv} = A_{vu} = w(u, v) = w(v, u)$ as a message sent from u to v incurs the same load as a message sent from v to u. Without loss of generality, we can re-normalize A_{uv} such that $0 \leq A_{uv} \leq 1$ by dividing each A_{uv} by the sum of all the entries in A. Thus, each normalized A_{uv} represents a percentage of the total communication load incurred by user u communicating to user v.
- $X \in \{0, 1\}^{M \times N}$: An assignment matrix where $X_{u,i} = 1$ if user u is assigned to server i, and $X_{u,i} = 0$ otherwise. Since a user is assigned exactly to one server, $\sum_{i=1}^{N} X_{u,i} = 1$.
- $L^{N \times N}(X)$: The server load matrix for an assignment X. $L_{ij}(X)$ represents the communication load from server i to server j. $L_{ii}(X)$ represents the local load caused by the communication among all the users assigned to server i only.
- $S(X)$ and $S_i(X)$: The total system load from all the servers and the load at each server i.
- $\overline{S}(X)$: The average server load, i.e., the total load from all the servers divided by the number of servers.
- The load imbalance at server i is defined as:

$$\Delta S_i(X) \equiv S_i(X) - \overline{S}(X), \tag{1}$$

  where $S_i(X)$ denotes the total load at server i.
- We now propose to minimize the following objective function:

$$F(X) = \alpha S(X) + (1 - \alpha) \max_i \Delta S_i(X), \tag{2}$$

  where α denotes a pre-specified weighted positive coefficient. α controls the trade-off parameter between total system load and load balance with $0 \leq \alpha \leq 1$.

Minimizing $F(X)$ results in reducing the total system load $S(X)$ while achieving the load balance among the server by reducing the maximum load imbalance of a server $\max_i \Delta S_i(X)$[1]. We assume that M, N, G, $A^{M \times M}$ are given. Our goal is to find the assignment matrix X that minimizes $F(X)$. Mathematically, the client-server assignment problem can be formulated as:

$$\begin{array}{ll} \textbf{Minimize:} & F(X) \\ \textbf{Subject to:} & X_{ui} \in \{0, 1\} \\ & \sum_i X_{ui} = 1 \\ & \forall u \in \{1, 2, .., M\}, i \in \{1, .., N\} \end{array} \tag{3}$$

Proposition 1: The optimization problem in (3) is NP-Hard.

Proof: Please see Appendix A. ■

To solve the problem above, we will first find the mathematical expressions for $L^{N \times N}(X)$, $S(X)$ and $S_i(X)$ and $F(X)$ in terms of X. We begin with the derivation of server load $L^{N \times N}(X)$. Note that $L(X)$ is symmetric due to our assumption that sending and receiving messages incur the same cost.

[1]There are several options to choose the imbalance factor. For example the Gini index [30] can be selected as the imbalance factor.

Now let us define the inter-server load between server i and $j \neq i$ as the load incurred by the communication between users assigned to server i and users assigned to server $j \neq i$. Since each entry of X is either 0 or 1, the inter-server load between servers i and j can be written as a function of the assignment matrix X as:

$$L_{ij}(X) = \sum_{k=1}^{M} \sum_{l=1}^{M} A_{kl} X_{ki} X_{lj}. \tag{4}$$

Define the local load of a server i as the load incurred only by the communication among the users assigned to the server i only. The local load of server i can be written as a function of the assignment matrix X as:

$$L_{ii}(X) = \frac{1}{2} \sum_{k=1}^{M} \sum_{l=1}^{M} A_{kl} X_{ki} X_{li}. \tag{5}$$

From (4) and (5), the server load matrix $L(X)$ can be compactly written as a function of the assignment matrix X as:

$$L(X) = X^T A X - \frac{1}{2} Diag(X^T A X), \tag{6}$$

where $Diag(.)$ denotes an operator on a matrix X that results in a diagonal matrix whose diagonal elements are the diagonal elements of X.

Since the total system load $S(X)$ for an assignment X is the sum of all the entries in $L(X)$, $S(X)$ can be written as:

$$S(X) = ||X^T A X||_1 - \frac{1}{2} Tr(X^T A X), \tag{7}$$

where $||.||_1$ denotes the entry-wise l_1-matrix norm, i.e., the sum of all the entries in the matrix.

As for the load balance metric, it is convenient to consider the opposite, i.e., the load imbalance. For this, we first define the average load of all the servers as:

$$\overline{S}(X) = \frac{S}{N} = \frac{||X^T A X||_1 - \frac{1}{2} Tr(X^T A X)}{N}. \tag{8}$$

Finally, $F(X)$ can be written in terms of the assignment matrix X as:

$$\begin{aligned} F(X) = {} & \alpha \frac{(N-1)}{N} (||X^T A X||_1 - \frac{1}{2} Tr(X^T A X)) \\ & + (1 - \alpha) ||(X^T A X - \frac{1}{2} Diag(X^T A X)) \mathbf{1}||_\infty, \end{aligned} \tag{9}$$

where $||x||_\infty$ is $\max_i |x_i|$, i.e., the maximum magnitude component in a vector x.

We note that minimizing $F(X)$ implies that minimizing the total load and the load imbalance. However, as previously discussed, to obtain the lowest total load (putting all the users in one server), it is necessary that the load imbalance must increase. Therefore, the tuning coefficient α allows for the trade-off between the total load and the load balance. At one extremity, setting $\alpha = 0$ implies minimizing the load balance regardless of the total load, while setting $\alpha = 1$ implying minimizing the total load regardless of the load balance.

Unfortunately, minimizing $F(X)$ over $X \in \{0,1\}^{M\times N}$ is hard. Therefore, we propose heuristic algorithms for solving the problem in a distributed approach such that global optimal solutions can be achieved with high probability.

A. ALGORITHMIC OUTLINE

We are interested in a *distributed* and iterative approach where servers exchange their "summarized" information among each other iteratively. Initially, clients are assigned uniformly at random to the servers. At each iteration, based on the exchanged information, a pair of servers then decide to move their users to each other appropriately. Furthermore, each server is assumed to keep the information about its assigned users and their immediate neighbors (adjacent vertices in G) only. No global information is allowed. For example, each server does not have the matrix A that represents the communication patterns among all the users, but only a portion of A. One advantage of this approach is that the algorithm can be viewed as a network protocol, running in real-time. Servers move their users accordingly at every time step. Over time, a good distributed algorithm will converge, i.e., no user is moved after some number of time steps, and the resulted assignment X produces approximately minimal $F(X)$ as defined in (9).

That said, the primary components our proposed distributed algorithm relies on:

1) What information is to be computed and kept at the servers?
2) What information is to be exchanged among the servers?
3) At every time step, how to decide which pairs of servers are involved in moving their users?
4) Within this chosen pair of servers, which users are to be moved?

Notation Augmentation. Before answering these questions, we first augment previously defined notations to include an index t that presents the iteration. This is necessary since our focus is on how the objective function changes with each iteration. For example, X becomes $X(t)$ to represent the assignment at time step t; $F(X)$ now becomes $F(X(t))$ which represents the value of the objective function at time t. For convenience, we also drop the variable X from all the functions; for example, $S(X(t))$ and $F(X(t))$ become $S(t)$ and $F(t)$. Next, we summarize a few theoretical results used in developing our algorithm.

B. TECHNICAL BUILDING BLOCKS

Since our proposed algorithm involves reassigning users at every iteration and a main component of the objective function is the total system load, we will consider how the total system load changes when a user u is moved from server i to server j.

At time t, if user u is moved from server i to server j, then there are changes in load for servers i and j only; the loads of other server remain the same. Specifically, load at servers at time $t+1$ will be:

- The new load in server i is:

$$S_i(t+1) = S_i(t) - \sum_{s=1|s\neq i}^{N} \sum_{l=1}^{M} A_{ul} X_{ls}(t) \tag{10}$$

- The new load in server j is:

$$S_j(t+1) = S_j(t) + \sum_{s=1|s\neq j}^{N} \sum_{l=1}^{M} A_{ul} X_{ls}(t) \tag{11}$$

- The load in any server $s \neq i, j$ does not change:

$$S_s(t+1) = S_s(t) \tag{12}$$

- Consequently, the new total system load is:

$$S(t+1) = S(t) + \sum_{l=1}^{M} A_{ul}(X_{li}(t) - X_{lj}(t)) \tag{13}$$

The first assertion in (10) is true because the term with the double sum represents the load that user u incurred on server i due to its communication with other users in other servers. Note that after u is moved from server i to server j, the amount of load incurred by u on server i due its communication with the users assigned to server i remain the same before and after u is moved. Therefore, i is excluded from the index in the outer sum in (10). The minus sign reflects the amount of load taken away from server i. By similar reasoning, the second assertion in (11) is true. The only difference is that the sign in the double sum term is positive, reflecting a load increase for server j. The third assertion in (12) is true because for a server $s \notin \{i, j\}$, the number of messages that it sends and receives, remain the same before and after the move. The fourth assertion in (13) is true by summing the changes in (10) and (11).

In addition to the total load, we also need to consider the load balance, or more precisely the system load imbalance. Consider again that a user u is moved from server i to server j and define:

$$\delta(t) = \sum_{l=1}^{M} A_{ul}(X_{li}(t) - X_{lj}(t)). \tag{14}$$

Intuitively, $\delta(t)$ is the change in the inter-server load. So when a user is moved from one server to another, only the inter-server load changes. From (13) and (14), we have:

$$S(t+1) = S(t) + \delta(t), \tag{15}$$

$$\overline{S}(t+1) = \frac{S(t+1)}{N} = \overline{S}(t) + \frac{\delta(t)}{N}. \tag{16}$$

The load imbalance at servers i and j are:

$$\Delta S_i(t+1) = \Delta S_i(t) - \sum_{k=1|k\neq i}^{N} \sum_{l=1}^{M} A_{ul} X_{lk}(t) - \frac{\delta(t)}{N} \tag{17}$$

$$\Delta S_j(t+1) = \Delta S_j(t) + \sum_{k=1|k\neq j}^{N} \sum_{l=1}^{M} A_{ul} X_{lk}(t) - \frac{\delta(t)}{N} \tag{18}$$

Now, the load imbalance at server $s \notin \{i, j\}$ is:

$$\begin{aligned}\Delta S_s(t+1) &= S_s(t+1) - \overline{S}(t+1)\\ &= \Delta S_s(t) - \frac{\delta(t)}{N}\end{aligned} \tag{19}$$

The value of the objective function at time $t+1$ is:

$$F(t+1) = \alpha S(t+1) + (1-\alpha)\max_i \Delta S_i(t+1). \tag{20}$$

The results above are used in subsequent algorithm to compute the new global objective function value at every iteration when a user is reassigned.

IV. ALGORITHMS

In this section, we introduce three algorithms: The centralized Simulated Annealing (CSA) algorithm, the Distributed Perturbed Greedy Search (DPGS) algorithm, and the Distributed Simulated Annealing (DSA) algorithm. All of these algorithms use the mathematical derivations from the preceding sections.

A. CENTRALIZED SIMULATED ANNEALING ALGORITHM

We describe the Centralized Simulated Annealing (CSA) algorithm for the client-server assignment problem. The performance of the CSA algorithm will be used as a baseline for evaluating distributed algorithms. The CSA algorithm is based on the simulated annealing (SA) framework which has been shown highly effective in solving many large scale combinatorial optimization problems [12]. The CSA algorithm is a type of stochastic greedy search in which the probability of searching in the next configuration is based on the objective value at the current and the next configuration. Central to the design of the CSA algorithm is the slowly decreasing temperature parameter *Tb*, which allows for the algorithm to explore many configurations before settling down to the approximately optimal configuration. Also, the probabilistic search allows the CSA algorithm to overcome the local minimums.

For the client-server assignment problem, the objective function is $F(X)$ and the configuration or state is the assignment matrix X. A centralized controller is dedicated to the optimization process. Users and their communication pattern are stored centrally at the controller. The controller execute the CSA algorithm to find an optimum assignment by minimizing the objective function as in (2) and clients will then be assigned to the servers according to the new configuration X. The pseudo codes for the CSA algorithm is shown in Algorithm 1.

Initially, the centralized controller computes the global parameters such as loads (7), imbalances (8) and objective function value (2) for some random assignment X. Starting at a random assignment and relatively high temperature *Tb*, the centralized controller selects a next assignment by selecting an arbitrary user u to change its server and computes corresponding objective function value by (20). If the new assignment has a lower objective function value, it will be selected as the new assignment; otherwise, it will be selected with a probability of $exp(-\frac{\Delta F}{Tb})$ where ΔF is the increasing amount in globally objective function. After each iteration, the temperature *Tb* is decreased if it is still above a threshold T_ϵ. The optimization process stops if *Tb* reaches a given freezing value ϵ.

Algorithm 1 Centralized SA Algorithm (CSA)

Calculate global parameters: $S, \overline{S}, \Delta S, F$
Initialize *Tb*
while $Tb > \epsilon$ **do**
  Select an user u
  Select a new server s
  Calculate EstimateF
  if EstimateF < F **then**
    Switch user u to new server s
  else
    Switch user u to new server s
    with probability $exp(\frac{F-EstimateF}{Tb})$
  end if
  if $Tb > T_\epsilon$ **then**
    Decrease *Tb*
  end if
end while

B. DISTRIBUTED DATA STRUCTURES AND COMPUTATIONS

In this section, we describe the distributed data structures and computations to be used by the two proposed distributed algorithms DSA and DPGS. Unlike a centralized algorithm that uses the global information such as the user communication pattern matrix A, distributed algorithms require that individual servers only employ local information. In our proposed distributed algorithms, at every time step, each server keeps track the following local information:

- $M^{(i)}(t)$: Number of users assigned to server i at time t.
- $V^{(i)}(t)$: Set of users assigned to server i at time t.
- $A^{(i)}(t)$: An $M^{(i)}(t) \times M^{(i)}(t)$ matrix represents the amount of load incurred at server i at time step t by the users assigned to server i only. Specifically, $A^i_{uv}(t)$ represents the frequency of user u sending messages to user v with both u and v belonging to server i.
- $I^{(i)}(t)$: An $M^{(i)} \times N$ matrix whose an entry $I^{(i)}_{us}(t)$ represents the amount of inter-server load incurred by user u in server i communicating with all the users in server s.
- $V^{(i)}_I$: Set of users assigned to server i that have friends on other servers.

It is important to note that often $A^{(i)}(t)$ is much smaller than A. This is due to the fact that the number of users per server is $O(M/N)$. Similarly, $I^{(i)}(t)$ is small. To illustrate $A^{(i)}(t)$ and $I^{(i)}(t)$, Fig. 3 shows an example of a communication pattern among 10 users that are assigned to 3 servers. The global matrix A representing the communication graph and the global assignment matrix X are depicted

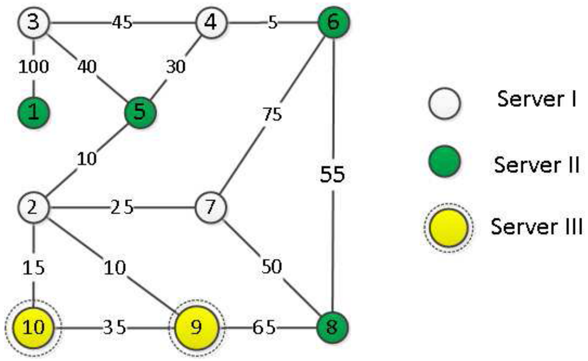


FIGURE 3. Example of user graph and 3-server assignment.

$$A = \begin{bmatrix} & & 100 & & & & & & & \\ & & & & 10 & & 25 & & 10 & 15 \\ 100 & & & 45 & 40 & & & & & \\ & & 45 & & 30 & 5 & & & & \\ & 10 & 40 & 30 & & & & & & \\ & & & 5 & & & 75 & 55 & & \\ & 25 & & & & 75 & & 50 & & \\ & & & & & 55 & 50 & & 65 & \\ & 10 & & & & & & 65 & & 35 \\ & 15 & & & & & & & 35 & \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

FIGURE 4. Global matrices of graph in Fig. 3.

$$A^{(1)} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccc} 2 & 3 & 4 & 7 \end{array} \\ \begin{bmatrix} & & & 25 \\ & & 45 & \\ & 45 & & \\ 25 & & & \end{bmatrix} \end{array} \qquad I^{(1)} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} \mathrm{II} & \mathrm{III} \end{array} \\ \begin{bmatrix} 10 & 25 \\ 140 & 0 \\ 35 & 0 \\ 125 & 0 \end{bmatrix} \end{array}$$

$$A^{(2)} = \begin{array}{c} \\ 1 \\ 5 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 5 & 6 & 8 \end{array} \\ \begin{bmatrix} & & & \\ & & & \\ & & & 55 \\ & & 55 & \end{bmatrix} \end{array} \qquad I^{(2)} = \begin{array}{c} \\ 1 \\ 5 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{cc} \mathrm{I} & \mathrm{III} \end{array} \\ \begin{bmatrix} 100 & 0 \\ 70 & 0 \\ 80 & 0 \\ 50 & 65 \end{bmatrix} \end{array}$$

$$A^{(3)} = \begin{array}{c} \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cc} 9 & 10 \end{array} \\ \begin{bmatrix} & 35 \\ 35 & \end{bmatrix} \end{array} \qquad I^{(3)} = \begin{array}{c} \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cc} \mathrm{I} & \mathrm{II} \end{array} \\ \begin{bmatrix} 10 & 65 \\ 15 & 0 \end{bmatrix} \end{array}$$

FIGURE 5. Local matrices at servers for assignment scheme in Fig. 3.

in Fig. 4. A centralized algorithm would have both A and X. This is not necessary in our proposed distributed setting where each server maintains only local corresponding matrices $A^{(i)}(t)$ and $I^{(i)}(t)$ as shown in Fig. 5. Often, they are small, easy to maintain, since each server only needs to keep track of its user activities. Using only $A^{(i)}(t)$ and $I^{(i)}(t)$, the total load of server i at time t can be computed as:

$$S_i = \sum_{s=1}^{N}\sum_{u=1}^{M^{(i)}} I_{u,s}^{(i)} + \sum_{u=1}^{M^{(i)}}\sum_{l=1}^{M^{(i)}} A_{u,l}^{(i)}(k). \tag{21}$$

The servers then exchange their total loads $S_i(t)$ among themselves at each time step. Thus, every server knows the load of all other servers at any time step. Based on this, any server i can compute the globally total load, $S = \sum_i S_i$, therefore average load, the load imbalance of every server, and the globally objective function value are also computed by using (1), (2) and (8). Finally, the objective function value is computed distributively at each server as follows:

$$\delta(t) = \sum_{l=1}^{M^{(i)}} A_{u,l}^{(i)}(t) - I_{u,j}^{(i)}(t) \tag{22}$$

$$\Delta S_i(t+1) = \Delta S_i(t) - \sum_{k=1,k\neq i}^{N} I_{u,k}^{(i)}(t) - \frac{\delta(t)}{N} \tag{23}$$

$$\Delta S_j(t+1) = \Delta S_j(t) + \sum_{k=1,k\neq j}^{N} I_{u,k}^{(i)}(t) - \frac{\delta(t)}{N} \tag{24}$$

$$\Delta S_s(t+1) = \Delta S_s(t) - \frac{\delta(t)}{N}. \tag{25}$$

It can be seen that the equations above are identical with those in (14), (17)–(19), respectively, where the global user matrix A and the assignment matrix X have been replaced by the local matrices. Thus, it is completely distributed. We are now ready to describe our first distributed algorithm: DSA.

C. DISTRIBUTED SIMULATED ALGORITHM (DSA)

In Section IV-A, we describe the CSA algorithm to be used for small systems in relatively static settings. In many practical systems, the number of users can be thousands or even millions. Thus, we propose a Distributed Simulated Annealing (DSA) algorithm as follows. Assume that each server has local information on its users as described in Section IV-B. Using the DSA algorithm, the servers keep track of their local information such as the total load, the inter-server communication loads, and exchange these information with all other servers. Once a server receives all information of other servers, it will compute the global parameters and be able to anticipate the change in the global objective function value for moving a user from one server to another. Just as in CSA algorithm, DSA algorithm moves one user from one server to another probabilistically according to the simulated annealing method. To perform the simulated annealing process in a distributed manner, at each iteration, a server is selected uniformly at random. Next, the chosen server selects its users and reassigns them to another server probabilistically based on local computation as in (22)–(25). The only difference between DSA from CSA is that for DSA, only users ($V^{(i)}$) belonging to the chosen server i can be moved while for CSA, any user can be moved. Consequently, the CSA algorithm has the potential to converge to a good assignment faster than the DSA algorithm since at each iteration, it has the potential to take a better move due to more options. This intuition is in fact confirmed by the simulation results in Section V. On the other hand, the DSA algorithm does not need a centralized controller to keep track of all the

Algorithm 2 Distributed SA Algorithm (DSA)

for Each server s **do**
  V_s = set of users being assigned to current server
  Estimate communication load
  Update local matrices: $A^{(s)}, I^{(s)}$
  Calculate local load: S_s
  Sends/Receives S_s/S_k to/from server $k, \forall k \neq s$
  Calculate global parameters: $S, \overline{S}, \Delta S, F$
end for
Initialize Tb
while $Tb > \epsilon$ **do**
  Select an active server i
  for each user $u \in V^{(i)} \setminus V_{new}^{(i)}$ **do**
    Select a new server $s \neq i$
    Calculate EstimateF
    if EstimateF < F **then**
      Switch user u to new server s
    else
      Switch user u to new server s
      with probability $exp(\frac{F - EstimateF}{Tb})$
    end if
  end for
  if $Tb > T_\epsilon$ **then**
    Decrease Tb
  end if
end while

global information. Algorithm 2 lists the pseudo code for the DSA algorithm.

D. DISTRIBUTED PERTURBED GREEDY SEARCH (DPGS)

In this section, we propose another distributed algorithm called Distributed Perturbed Greedy Search (DPGS) with better performance using the two phases: the greedy search phase and the perturbation phase. The greedy search phase is a deterministic procedure that allows for the algorithm to reach a local optimal assignment quickly. Once a local optimal assignment is reached, the perturbation phase begins. The perturbation phase is a probabilistic procedure that moves a small number of users from one server to another to prevent the algorithm from getting stuck in the local minimum. The process then repeats again with the greedy and perturbation phases alternating until a good solution is obtained. In this sense, the greedy phase allows the algorithms finds a good starting point quickly, and the perturbation phase avoids local minimum similar to the standard simulated annealing process. We note that the simulated annealing process is done at every iteration which can take a long time to reach a good solution due to a large number of random search directions. On the other hand, in the proposed DPGS, the perturbation is only performed when a local minimum is reached, thus the search is more directed. In addition, instead of selecting the servers uniformly at random as is done in the DSA algorithm, the DPGS algorithm chooses the most overloaded

Algorithm 3 Distributed Perturbed Greedy Search (DPGS)

for Each server s **do**
  V_s = set of users being assigned to current server
  Estimate communication load
  Update local matrices: $A^{(s)}, I^{(s)}$
  Calculate local load: S_s
  Sends/Receives S_s/S_k to/from server $k, \forall k \neq s$
  Calculate global parameters: $S, \overline{S}, \Delta S, F$
end for
while $Tb > \epsilon$ **do**
  for each server i **do**
    Execute Algorithm 4
  end for
  if No moves have been made **then**
    Execute Algorithm 5
  end if
  Sends/Receives S_s/S_k to/from server $k, \forall k \neq s$
  if $Tb > T_\epsilon$ **then**
    Decrease Tb
  end if
end while

server for moving its users. We now elaborate on the two phases:

Choosing user in the greedy phase.

Proposition 2: If user u who is being assigned to server i has no friend on other servers, the global objective function will not decrease if u is reassigned to a new server.

Proof: We have

$$A_{ul}X_{lj} = 0 \qquad \forall j \neq i$$

so

$$\begin{aligned}
\delta(t) &= \sum_{l=1}^{M} A_{ul}X_{li} > 0 \\
S(t+1) &= S(t) + \delta(t) \\
\Delta S_s(t+1) &= \Delta S_s(t) - \frac{\delta}{N} \\
\Delta S_i(t+1) &= \Delta S_i(t) - \frac{\delta}{N} \\
\Delta S_j(t+1) &= \Delta S_j(t) + \delta(t) - \frac{\delta}{N}
\end{aligned}$$

Therefore

$$\begin{aligned}
F(t+1) &= S(t+1) \\
&\quad + \max\{\Delta S_i(t+1), \Delta S_j(t+1), \Delta S_s(t+1)\} \\
&> F(t) + \delta(t) - \min\{\frac{\delta(t)}{N}, \delta(t) - \frac{\delta(t)}{N}\} \\
&> F(t) \qquad \blacksquare
\end{aligned}$$

A direct result of this proposition is that only users who have friends being assigned to different servers will be considered to be reassigned in the greedy phase of Algorithm 3

Algorithm 4 Greedy Phase

1: **if** i is an overloaded server **then**
2:   **for** each u in $V_I^{(i)}$ **do**
3:     **for** $s = 1$ to N, $s \neq i$ **do**
4:       Calculate $EstimateF$ by (22)
5:       **if** $EstimateF < F$ **then**
6:         Reassign user u to new server s
7:         Update F
8:       **end if**
9:     **end for**
10:   **end for**
11:   Send updated S_i to other servers
12: **end if**

Algorithm 5 Perturbation Phase

1: **for** each $u \in V^{(i)} \setminus V_{new}^{(i)}$ **do**
2:   Select a new server $s \neq i$
3:   Calculate EstimateF
4:   **if** $EstimateF < F$ **then**
5:     Switch user u to new server s
6:   **else**
7:     Switch user u to new server s
8:     with probability $exp(\frac{F-EstimateF}{Tb})$
9:   **end if**
10: **end for**

(step 2 of Algorithm 4). This helps reducing the computations by skipping users who only have local communication. Especially, in a good assignment, most friendships are located on the same server.

Algorithm 4 shows the pseudo code for the DPGS algorithm. Initially, each server computes its local information: total load, inter-server communication loads, and exchanges with all other servers. Once a server receives all information of other servers, it will be able to compute the global parameters and decide whether it is an overloaded server. If it is an overloaded server, it will execute the greedy phase of the algorithm where it selects an user which has friends being assigned to different servers and computes gain if the chosen user will be reassigned to another server. Server only reassigns its user(s) to another server if it achieves the gain in the global objective function value (Algorithm 4). This process will lead the system to a local optimal assignment where no further reassignment can be made in order to achieve gain. Next, the algorithm changes to the perturbation phase. In the perturbation phase, all servers move users probabilistically similarly to a single iteration in the CSA algorithm (Algorithm 5).

All servers exchange updated information after each iteration. The temperature Tb is used to control the rate of moving users in the perturbation phase. The threshold values T_ϵ and ϵ also determine whether the configuration of the system is freezed or keep running in order to make the system adapt with the changes in the communication pattern of users.

E. CONVERGENCE ANALYSIS OF DPGS

In this section, we present the convergence analysis of the DPGS algorithm. There are two phases in the proposed DPGS algorithm: the greedy and perturbation phases. The perturbation phase is similar to simulated annealing process whose convergence rate is proved [31]. In this section, we concentrate on the convergence of the greedy phase.

We first define a quantity that will be used for the convergence analysis.

Definition 1: Let $\mathrm{w}(A)$ be the overall system load if all users are assigned to a single server.

$$\mathrm{w}(A) = \sum_{u=2}^{M}\sum_{l=1}^{u} A[u,l] = \frac{1}{2}1^T A 1 = \frac{1}{2}\sum_{v \in V} deg(v) \quad (26)$$

To bound the number of steps to the convergence of the greedy phase, we need two following propositions:

Proposition 3: Bounds on the objective function

$$\alpha \mathrm{w}(A) \leq F(X) \leq 2\alpha \mathrm{w}(A) + (1-\alpha)\frac{N-1}{N}\mathrm{w}(A) \quad (27)$$

Proof: See Appendix B. ∎

Intuitively, the lower bound can be reached when the graph of user communication patterns has connected components and can be assigned such that any connected component is assigned to a single server and all servers are perfectly balanced. The first term in the upper bound is the global load of the system when every friendship is split into different servers; the second term is the global imbalance when all users are assigned into a single server.

Proposition 4: If user u being assigned to server i at time t (an overload server) switches to server j at time $t+1$ if and only if $F(t+1) < F(t)$, the worst gain in global objective value is

$$\Delta F(t) = -\frac{\alpha N - (1-\alpha)}{N} m \quad (28)$$

where m is the smallest non-zero entry in A.

Proof: See Appendix C. ∎

Convergence. At every iteration of the greedy phase, the users are moved to another servers if and only if the value of the global objective function $F(X)$ decreases. This fact together with the Proposition 3 which shows that $F(X)$ has a lower bound, show that the Algorithm 4 always converges. The proposition below shows the maximum number of time steps for greedy phase to converge to a local minimum.

Proposition 5: Denote T as the number of time steps until the algorithm converges in greedy phase, we have:

$$T \leq \frac{F_0 - \alpha w(A)}{\frac{\alpha N - (1-\alpha)}{N} m}, \quad (29)$$

where F_0 is the value of the objective function at the beginning of each (distributed) optimization iteration.

Proof: The maximum number of steps in the greedy phase is equal to the maximum accumulated gain divided by the minimum gain per iteration step. The numerator is the upper bound of the accumulated gain which is the difference between the upper and lower bounds of the objective function shown in Proposition 3. The denominator is the minimum gain per iteration which is stated in the Proposition 4. ∎

We note that the convergence rate depends on the initial assignment, and importantly on the characteristics of user communication patterns matrix A.

Non-stopping optimization and adaptiveness to changes. In many scenarios, it is preferable to continuously optimize in order to respond to the environmental dynamics. In general, the design of the simulated annealing algorithm will ensure the algorithm convergence due to the decreasing of the temperature parameter *Tb* to zero (freezing). To avoid freezing so that the system can adapt to network dynamics, we ensure that *Tb* will never goes below certain temperature threshold T_ϵ. Choosing an appropriate temperature is an important matter in order to help system operate optimally. One suggested value is:

$$T_\epsilon = \mu \frac{F - F_L}{F_U - F_L}$$

where F are the current global objective function value, F_U, F_L are its upper bound and lower bound in Proposition 3 respectively; and $\mu \in (0, 1)$ is an empirical coefficient depending on user communication pattern and the changing rate. With such design, the DPGS algorithm is highly adaptive to the dynamic changes of the systems. Since each server keeps updating local load information according to any change such as adding/removing links, users and even servers. These changes are automatically reflected in local matrices such as $A^{(i)}$, $I^{(i)}$ and their values are used in the successive optimization iterations no matter whether the system is currently in the greedy or perturbation phase.

Complexity. Each server needs to store matrices $A^{(i)}$, $I^{(i)}$. On average, each server has an approximate number of users of $\frac{M}{N}$, the expectation of space complexity will be $O((M/N)^2)$. At each time step, the most overloaded server tries moving users to other servers until no more gain is achieved, so the time complexity for each step is $O(MN)$.

V. SIMULATION RESULTS

In this section, we present simulation results and compare three proposed algorithms. The evaluations were done with two types of data sets: synthetic networks and real-world online social networks. In all our simulations, we set $\alpha = 0.5$, allowing equal weight for load balance and total load.

A. SMALL SYNTHETIC GRAPHS

First, we test the distributed optimization algorithms with small synthetic graphs and compare solutions with exhaustive search results. Simulations were done for random graphs, regular graphs, and power-law graphs which were generated using the Barabasi's algorithm. We note that many types of social networks have been observed to follow power-law graphs [32].

Entries of A are generated uniformly at random between 0 and 1, and then normalized by the sum of all the entries in A. To determine the solution quality of an algorithm, we define the optimality as $\frac{F_{worst}-F}{F_{worst}-F_{best}}$, where F is the optimal value produced by the algorithm, F_{worst} and F_{best} correspond to the worst and the best assignments obtained from the exhaustive search algorithm on a given graph. A larger optimality indicates a better solution, e.g., $\frac{F_{worst}-F}{F_{worst}-F_{best}} = 1$ implies that the algorithm always obtains the best solution. Unfortunately, the exhaustive search is only feasible for small problem instances since our problem is *NP*-hard. Therefore, in Fig. 6a, we show a small problem instance where the DPGS algorithm found the same value of the global optimal value for a graph of 16 users assigned to 3 servers ($F = F_{best} = 0.6375$, $Opt. = 100\%$, $S_{max}/S_{min} = 1.3205$, $S_I/S = 0.0919$). Fig. 6(b) shows an example of convergence of the global function objective value. The local optimal values are reflected in steady region. After some small perturbations, the DPGS algorithm is able to escape from the local point (which might be a greater value) and the system continues the optimization process.

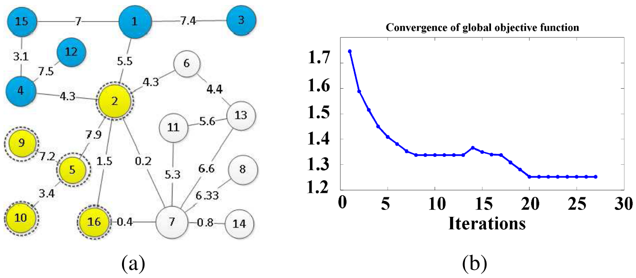


FIGURE 6. An example of an optimal assignment and the convergence rate of the proposed algorithm. (a) Assignment result. (b) Convergence of F.

We examine the average solution quality of the proposed algorithm and a random partitioner. To do so, we generate 100 graphs of each type at random. We applied the two algorithms to find the solutions, then averaged all solutions and compare to the results from the exhaustive search. Table 1 and Table 2 show the average optimality, the average objective function, the average ratio of the highest load to the lowest load of servers (S_{max}/S_{min}), and the average ratio of total inter-server load to the total server load for cases when (M,N) = (20,3) and (30,2) respectively. Two algorithms are compared by the values of F and S_I/S.

As seen in Tables 1 and 2, our proposed distributed algorithm gives better solutions consistently over the equal assignment algorithm in term of F, and other metrics. The proposed algorithm also approximates the optimal solutions very well as seen by the optimality ranging from 97% to 100% for the case $(M, N) = (20, 3)$, and 92% to 97% for the case $(M, N) = (30, 2)$.

B. LARGE GRAPHS

For larger size problems, we can only rely on the performance comparison among algorithms. Simulations are performed on two social network datasets: Facebook [33], BlogCatalog [34], and a synthetic network generated by Barabasi-Albert model [35]. The properties of these graphs which include centralizability, distributability, dynamical

TABLE 1. Solution quality metrics for (M,N) = (30, 2).

Graph type	F_{worst}	Best values			DPGS				Random partitioner				Improvement	
		F_{best}	$\frac{S_{max}}{S_{min}}$	$\frac{S_I}{S}$	F	Opt.	$\frac{S_{max}}{S_{min}}$	$\frac{S_I}{S}$	F	Opt.	$\frac{S_{max}}{S_{min}}$	$\frac{S_I}{S}$	F	$\frac{S_I}{S}$
Power-law	0.9934	0.5917	1.0066	0.0768	0.6009	97.73%	1.0343	0.0804	0.8021	47.62%	1.3775	0.3311	25.09%	75.72%
Regular	0.8500	0.6834	1.0000	0.1342	0.6834	100%	1.0000	0.1342	0.7856	38.64%	1.2168	0.3340	13.02%	59.82%
Random	0.9207	0.6282	1.0104	0.0907	0.6364	97.23%	1.0422	0.1030	0.7899	44.72%	1.2778	0.3314	19.44%	68.92%

TABLE 2. Solution quality metrics for (M,N) = (20, 3).

Graph type	F_{worst}	Best values			DPGS				Random partitioner				Improvement	
		F_{best}	$\frac{S_{max}}{S_{min}}$	$\frac{S_I}{S}$	F	Opt.	$\frac{S_{max}}{S_{min}}$	$\frac{S_I}{S}$	F	Opt.	$\frac{S_{max}}{S_{min}}$	$\frac{S_I}{S}$	F	$\frac{S_I}{S}$
Power-law	1.1642	0.6803	1.0812	0.1279	0.6936	97.27%	1.0132	0.1386	0.8030	74.65%	1.3779	0.3314	13.62%	58.18%
Regular	1.0351	0.7179	1.0112	0.1521	0.7429	92.10%	1.0555	0.1353	0.9065	40.55%	1.8934	0.3989	18.04%	66.08%
Random	1.0860	0.7218	1.0795	.14947	0.7377	95.64%	1.0684	0.1584	0.9140	47.22%	1.2489	0.4003	19.29%	60.43%

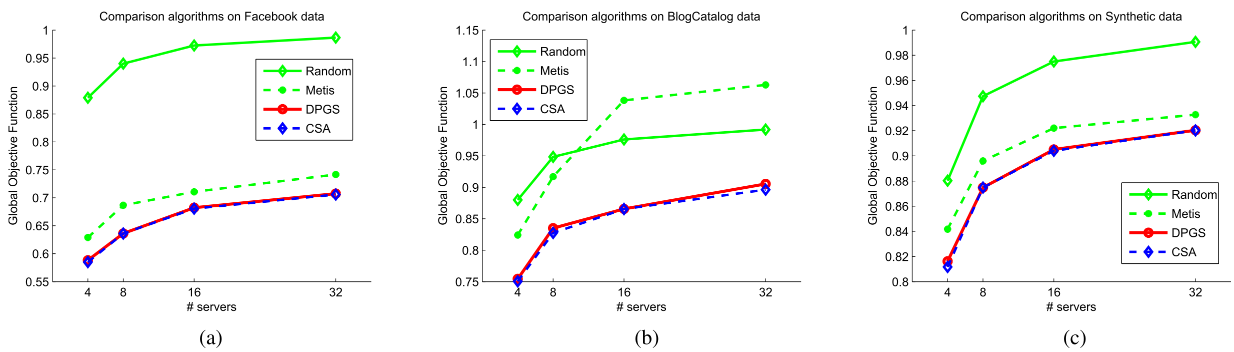


FIGURE 7. Performance comparison among algorithms. (a) Facebook. (b) BlogCatalog. (c) Synthetic.

adaptability, space complexity and time complexity, are summarized in Table 3.

TABLE 3. List of datasets.

	#nodes	#edges	Avg. deg.	Max deg.
Facebook[33]	63,731	817,090	25.64	1,098
BlogCatalog[34]	88,784	2,093,195	47.15	9,444
Synthetic	25,000	394,743	31.58	4,432

TABLE 4. Algorithm comparison.

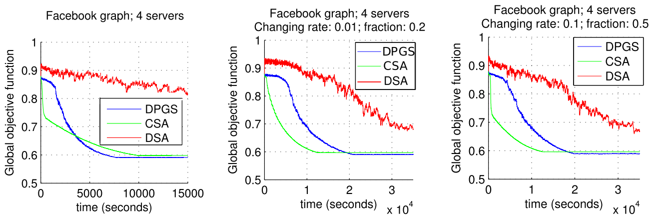
	Centr.	Distr.	Adapt.	Space Compl.	Time Compl.
Hashing	yes	yes	no	O(M)	O(M)
Metis	yes	no	no	O(M^2)	O($(M+\|E\|)log(N)$)
DSA	no	yes	yes	O($(\frac{M}{N})^2$)	O(MN)
CSA	yes	no	yes	O(M^2)	O(MN)
DPGS	no	yes	yes	O($(\frac{M}{N})^2$)	O(MN)

We compare our proposed algorithms with the random partitioner and Metis - a multilevel graph partitioning which runs fast and produces high quality partition for large graphs [36]. The random partitioner implements consistent hashing algorithms [26] which is used widely in practical system such as Cassandra [1]. Since $M >> N$, consistent hashing algorithm often produces equal size partitions. Table 4 compares the theoretical properties of our proposed algorithms with the other two algorithms.

Figure 7 shows the performance comparison of our proposed algorithm DPGS and the other two algorithms. Since DSA and CSA will converge to the same result, we just show the performance comparison between DPGS, CSA, random partinioner and Metis. It can be seen that DPGS and CSA produce the best performance among evaluated algorithms. In convergence mode, DPGS slightly perturbes the result hence it might produce a little worse result compared to CSA.

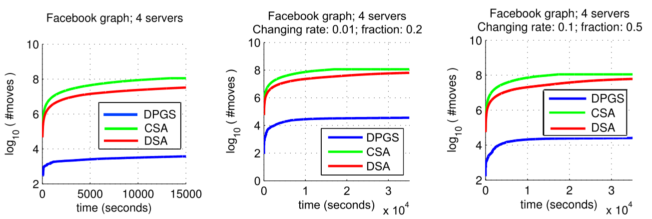
To test the effect of network dynamics on the algorithms, we initialize graphs with a random edge weights, and then we change the edge weights over time so that it will converge to a stationary graph structure which is the same as the graph in the 'no change' case. The network dynamics are reflected by the weight changing rate E and the fraction of graph edge changes F. For example, $(E = 0.01, F = 0.2)$ means 20% of edges change their weights at the rate of 1% after each iteration until the overall graph converges to the finally stationary graph.

Figures 8 and 10 show convergence behaviors of proposed algorithms for Facebook data with $N = 4$ and $N = 8$ servers respectively. The convergence of the objective function of different algorithms on the synthetic graph are shown on Figures 12 and 14 respectively. The high fluctuations at the



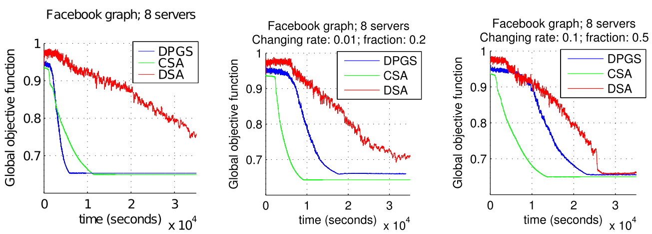
(a) (b) (c)

FIGURE 8. Objective functions values; Facebook data with $N = 4$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



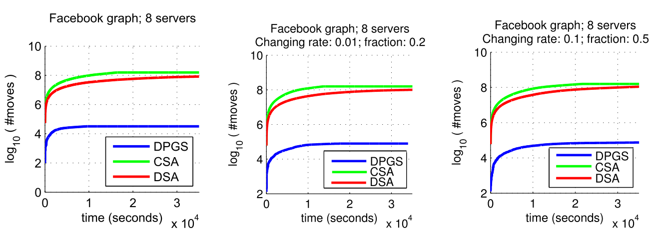
(a) (b) (c)

FIGURE 9. Number of moves; Facebook data with $N = 4$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



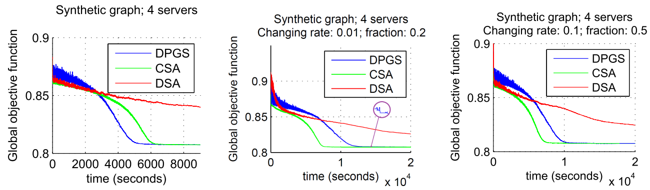
(a) (b) (c)

FIGURE 10. Objective functions values; Facebook data with $N = 8$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



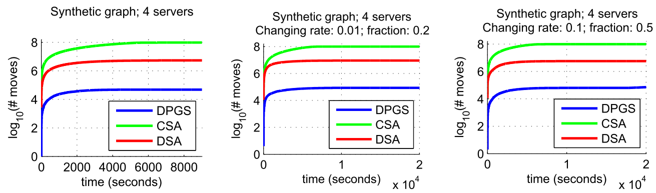
(a) (b) (c)

FIGURE 11. Number of moves; Facebook data with $N = 8$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



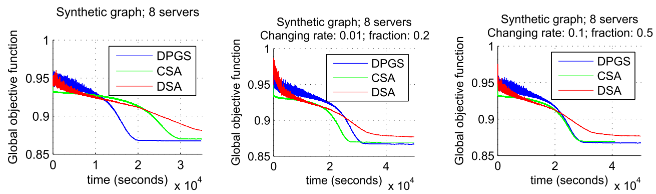
(a) (b) (c)

FIGURE 12. Objective functions values; Synthetic data with $N = 4$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



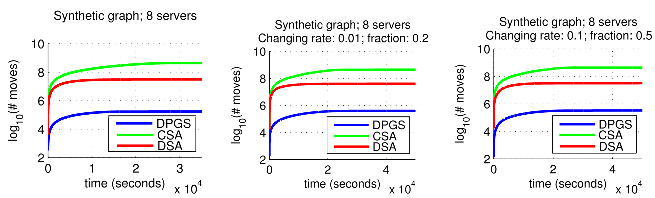
(a) (b) (c)

FIGURE 13. Number of moves; Synthetic data with $N = 4$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



(a) (b) (c)

FIGURE 14. Objective functions values; Synthetic data with $N = 8$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.



(a) (b) (c)

FIGURE 15. Number of moves; Synthetic data with $N = 8$ servers. (a) No change. (b) $E = 0.01$; $F = 0.2$. (c) $E = 0.1$; $F = 0.5$.

early stages are the effect of high temperature controlling the perturbation rates. The curves become smoother over time because the controlling temperatures decrease. It can be seen that the DPGS, which is feasible for many practical systems, performs quite well compared to CSA algorithm and outperforms the DSA algorithm in term of convergence rates.

Figure 9, Figure 11, Figure 13, Figure 15 compare the overheads of the three algorithms. The overhead of an algorithm is the total number of moves (user reassignments) that each algorithm has to make in order to evolve the system to the optimal state. The figures show the log_{10} of the total moves made by algorithms. As seen, the DPGS algorithm results in a substantially fewer number of reassignments than the other two algorithms.

Figure 8(b) and Figure 12(b) also show the effectiveness of letting the system remain at a low 'temperature' rather than stopping after reaching freezing temperature. The DPGS algorithm allows the system to adapt to the network dynamics (due to changes in weights of user graph) and to adjust accordingly so that it always stays at an optimal configuration. This property is reflected in the magnified segments of the curves in the stable regions.

VI. CONCLUSION

In this paper we proposed three heuristic algorithms for solving the client-server assignment problem: 1) The Centralized Simulated Annealing (CSA) algorithm, 2) the Distributed Simulated Annealing (DSA) algorithm, and 3) the Distributed Perturbed Greedy Search (DPGS). The CSA algorithm produces good solution in the fastest time, however it relies on timely accurate global system information, and is practical only for small and static systems. In contrast, the two distributed algorithms DSA and DPGS exploit local information at each server during their search for the optimal assignment, and thus can scale well with the number of users and servers as well as adapting to the system dynamics. Simulation results show that the performance of the distributed algorithms, specifically the DPGS algorithm, is competitive with that of the centralized algorithm while providing the advantage of naturally adapting to time-varying communication pattern of users. Theoretical results on convergence rate are given.

REFERENCES

[1] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[2] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Autom. Conf.*, Jun. 1982, pp. 175–181.

[3] J. Y. Zien, M. D. F. Schlag, and P. K. Chan, "Multilevel spectral hypergraph partitioning with arbitrary vertex sizes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1389–1399, Sep. 1999.

[4] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, "Spectral K-way ratio-cut partitioning and clustering," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 9, pp. 1088–1096, Sep. 1994.

[5] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[6] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 11, pp. 1101–1113, Nov. 1993.

[7] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Softw. Eng.*, vol. SE-3, no. 1, pp. 85–93, Jan. 1977.

[8] G. Sabin, V. Sahasrabudhe, and P. Sadayappan, "On fairness in distributed job scheduling across multiple sites," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2004, pp. 35–44.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

[10] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.

[11] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation. Part I, graph partitioning," *Oper. Res.*, vol. 37, no. 6, pp. 865–892, 1989.

[12] S. Kirkpatrick and M. P. Vecchi, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[13] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation. Part I, graph partitioning," *Oper. Res.*, vol. 37, no. 6, pp. 865–892, 1989.

[14] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 841–855, Jul. 1996.

[15] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the traveling salesman problem," *Complex Syst.*, vol. 5, pp. 299–326, Jan. 1991.

[16] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci. USA*, vol. 103, no. 23, pp. 8577–8582, 2006.

[17] Y. Saab, "A new effective and efficient multi-level partitioning algorithm," in *Proc. Conf. Design, Autom. Test Eur.*, 2000, pp. 112–116.

[18] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007.

[19] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, no. 5, pp. 1299–1319, 1998.

[20] 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 *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Dec. 2001, pp. 107–114.

[21] D. P. Vidyarthi, B. K. Sarker, A. K. Tripathi, and L. T. Yang, *Scheduling in Distributed Computing Systems: Analysis, Design and Models*. Berlin, Germany: Springer Science & Business Media, 2008. [Online]. Available: http://books.google.com/books?id=HqF-OzNstX4C

[22] Y. Jiang, Y. Zhou, and W. Wang, "Task allocation for undependable multiagent systems in social networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1671–1681, Aug. 2013.

[23] Y. Jiang and Z. Huang, "The rich get richer: Preferential attachment in the task allocation of cooperative networked multiagent systems with resource caching," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 5, pp. 1040–1052, Sep. 2012.

[24] Y. Jiang and J. Jiang, "Contextual resource negotiation-based task allocation and load balancing in complex software systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 641–653, May 2009. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2008.133

[25] Cassandra Wiki. *Random Partitioner*. [Online]. Available: http://wiki.apache.org/cassandra/RandomPartitioner, accessed Nov. 2013.

[26] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proc. 29th Annu. ACM Symp. Theory Comput. (STOC)*, 1997, pp. 654–663. [Online]. Available: http://doi.acm.org/10.1145/258533.258660

[27] H. Nishida and T. Nguyen, "Optimal client-server assignment for internet distributed systems," in *Proc. 20th ICCCN*, Jul./Aug. 2011, pp. 1–6.

[28] D. A. Tran, K. Nguyen, and C. Pham, "S-CLONE: Socially-aware data replication for social networks," *Comput. Netw.*, vol. 56, no. 7, pp. 2001–2013, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.02.010

[29] K. Nguyen, C. Pham, D. A. Tran, and F. Zhang, "Preserving social locality in data replication for social networks," in *Proc. IEEE ICDCS Workshop Simplifying Complex Netw. Pract. (SIMPLEX)*, Minneapolis, MN, USA, Jun. 2011, pp. 129–133.

[30] C. Gini, "Variabilità e mutabilità," *J. Roy. Statist. Soc.*, vol. 76, no. 3, pp. 326–327, Feb. 1913.

[31] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," in *Proc. 24th IEEE Conf. Decision Control*, vol. 24. Dec. 1985, pp. 761–767.

[32] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 251–262, Aug. 1999. [Online]. Available: http://doi.acm.org/10.1145/316194.316229

[33] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. 2nd ACM Workshop Online Soc. Netw.*, 2009, pp. 37–42.

[34] R. Zafarani and H. Liu. (2009). *Social Computing Data Repository at ASU*. [Online]. Available: http://socialcomputing.asu.edu

[35] A.-L. Barabási, R. Albert, and H. Jeong, "Scale-free characteristics of random networks: The topology of the world-wide web," *Phys. A, Statist. Mech. Appl.*, vol. 281, nos. 1–4, pp. 69–77, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378437100000182

[36] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

THUAN DUONG-BA received the B.S. and M.S. degrees in electronics and telecommunications from the Hanoi University of Science and Technology (HUST), Hanoi, Vietnam, in 2002 and 2005, respectively. From 2006 to 2009, he was with HUST as a Lecturer. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Oregon State University, Corvallis, OR, USA. His research interests include computer networks, network coding, random optimization, and distributed systems.

THINH NGUYEN received the B.S. degree from the University of Washington, Seattle, WA, USA, in 1995, and the Ph.D. degree from the University of California at Berkeley, Berkeley, CA, USA, in 2003. He is currently an Associate Professor with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA. He is interested in all things stochastic with applications to signal processing, distributed systems, wireless networks, network coding, and quantum walks. He served as an Associate Editor of the *IEEE Transactions on Circuits and Systems for Video Technology* and the *IEEE Transactions on Multimedia*.

BELLA BOSE (F'95) received the B.E. degree in electrical engineering from the University of Madras, Chennai, India, in 1973, the M.E. degree in electrical engineering from the Indian Institute of Science, Bangalore, India, in 1975, and the M.S. and Ph.D. degrees in computer science and engineering from Southern Methodist University, Dallas, TX, USA, in 1979 and 1980, respectively. Since 1980, he has been with Oregon State University, Corvallis, OR, USA, where he is currently a Professor and the Associate Director of the School of Electrical Engineering and Computer Science. His current research interests include error control codes, fault-tolerant computing, parallel processing, and computer networks. He is a fellow of the Association for Computing Machinery.

DUC A. TRAN (M'03) is currently a tenured Associate Professor of Computer Science with the University of Massachusetts at Boston, Boston, MA, USA, where he is also the Director of the Network Information Systems Laboratory. He received the Ph.D. degree in computer science from the University of Central Florida, Orlando, FL, USA, and the B.S. degree in computer science from Vietnam National University, Hanoi, Vietnam. His interests are focused on data-centric and networking designs targeting social, sensing, storage, and search applications. He has received several research funding awards from the National Science Foundation (NSF) and Best Paper Recognition at ICCCN 2008 and DaWak 1999. He has served several times as a Review Panelist of NSF, an Editor of the *Journal of Computational Social Networks* (since 2013), the *Journal of Parallel, Emergent, and Distributed Systems* (since 2010), and the *ISRN Communications and Networking* journal (since 2010), a Guest Editor of the *Journal of Pervasive Computing and Communications* (2009), the TPC Chair of WiMAN 2014, CCNet (2010 and 2011), GridPeer (2009, 2010, and 2011), and IRSN 2009, and the TPC Vice Chair of AINA 2007. He was a keynote speaker at the 2013 Workshop on Wireless Mesh and Ad Hoc Networks. He is an Organizing Committee Member of the 2014 ACM Multimedia Conference, serving as the History Preservation Chair. He is a Senior Member of the Association for Computing Machinery.