Optimizing Client Assignment for Enhancing Interactivity in Distributed Interactive Applications

Lu Zhang *Student Member, IEEE,* and Xueyan Tang *Senior Member, IEEE*

***Abstract*—Distributed Interactive Applications (DIAs) are networked systems that allow multiple participants at different locations to interact with each other. Wide spreads of client locations in large-scale DIAs often require geographical distribution of servers to meet the latency requirements of the applications. In the distributed server architecture, the network latencies involved in the interactions between clients are directly affected by how the clients are assigned to the servers. In this paper, we focus on the problem of assigning clients to appropriate servers in DIAs to enhance their interactivity. We formulate the problem as a combinational optimization problem and prove that it is NP-complete. Then, we propose several heuristic algorithms for fast computation of good client assignments and theoretically analyze their approximation ratios. The proposed algorithms are also experimentally evaluated with real Internet latency data. The results show that the proposed algorithms are efficient and effective in reducing the interaction time between clients, and our proposed *Distributed-Modify-Assignment* adapts well to the dynamics of client participation and network conditions. For the special case of tree network topologies, we develop a polynomial-time algorithm to compute the optimal client assignment.**

***Index Terms*—distributed interactive application, client assignment, interactivity, NP-complete.**

I. Introduction

DISTRIBUTED Interactive Applications (DIAs) are networked systems that allow multiple participants at different locations to interact with each other in real time. DIAs spread over a wide range of areas that are gaining popularity rapidly, such as multiplayer online games [2], distributed interactive simulations [3], and collaborative computer-aided design and engineering [4]. Typically, the application's state (such as the virtual worlds in multiplayer online games and the shared workspaces in collaborative design tools) is maintained by servers. Participants, known as clients, are responsible for sending user-initiated operations to the servers and receiving updates of the application's state from the servers. Since DIAs are human-in-the-loop applications, it is of crucial importance to improve the interactivity of DIAs for supporting graceful interactions among clients. The major challenges to interactivity enhancement are to deal with network latencies. Wide spreads of client locations in large-scale DIAs often require geographical distribution of servers to meet the latency requirements of the applications. This kind of latency-driven distribution is essential even when there are no limitations on the availability of server resources in one location [5].

In the distributed server architecture, the servers communicate with each other directly, while each client is assigned to one server and interacts with other clients through their assigned servers [6], [7]. The interactivity performance of DIAs can be characterized by the duration from the time when a client issues an operation to the time when the resultant state update is presented to the same client or a different client [8]. This duration shall be called the *interaction time* between these clients. In the distributed server architecture, the interaction time between any pair of clients includes the network latencies between the clients and their assigned servers, and the network latency between their assigned servers. These network latencies are directly affected by how the clients are assigned to the servers. Therefore, client assignment is an important issue for enhancing the interactivity of DIAs.

Finding the optimal client assignment is a challenging task. An intuitive approach is to assign each client to its nearest server [2], [9]. While this assignment reduces the client-to-server latency, it may considerably increase the latency between the assigned servers of different clients and thus perform far worse than optimum, as shall be shown by our experimental results. On the other hand, assigning all clients to a single server eliminates the contribution of the inter-server latency to the interaction time, but such assignment may significantly increase the latency between the clients and their assigned server, defeating the purpose of geographical distribution of servers. An optimal assignment for maximizing the interactivity performance should strike a balance between the inter-server latency and the client-to-server latency.

In this paper, we investigate the client assignment problem for enhancing the interactivity performance of DIAs. We formulate the problem as a combinational optimization problem on graphs and prove that it is NP-complete. Several heuristic assignment algorithms are proposed and their approximation ratios are analyzed. The algorithms are also evaluated by simulation experiments with real Internet latency data. The results show that the proposed algorithms are efficient and effective in reducing the interaction time between clients, and our proposed *Distributed-Modify-Assignment* adapts well to the dynamics of client participation and network conditions. For the special case of tree network topologies, we show that there exist a polynomial-time optimal solution, and develop an efficient algorithm to compute the optimal client assignment.

The rest of this paper is organized as follows. Section II formulates the client assignment problem. Section III proves the NP-completeness results. Section IV proposes three heuristic assignment algorithms and analyzes their approximation ratios. The experimental setup and results are discussed in Section V. Section VI presents a polynomial-time optimal solution for tree network topologies. Section VII summarizes the related work. Finally, Section VIII concludes the paper.

L. Zhang and X. Tang are with the School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore, 639798. E-mail: {zh0007lu, asxytang}@ntu.edu.sg. A preliminary report of this work is presented at IEEE INFOCOM 2011 conference [1].

II. Problem Formulation

We model the underlying network supporting the DIA by a graph $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of links between the nodes. A length $d(u, v) > 0$ is associated with each link $(u, v) \in E$, representing the network latency between nodes u and v. If a message transfer from one node to another goes through multiple links, the total latency is given by the sum of those on all intermediate links along the route. To facilitate presentation, we shall extend the function $d(u, v)$ to all pairs of nodes $(u, v) \in V \times V$ by defining $d(u, v)$ as the length of the routing path between nodes u and v. We also define that $d(u, v) = 0$ if $u = v$.

We assume a distributed server architecture. Let $C \subseteq V$ be the set of clients in the network and $S \subseteq V$ be the set of servers in the network. Each client is assigned to a server in order to send operations and receive state updates. An assignment A is a mapping from C to S, where for each client $c \in C$, we denote by $s_A(c) \in S$ the server that client c is assigned to. In this paper, we focus on reducing the network latency involved in the interaction between clients, since the network latency is generally more difficult to improve than the server-side processing delay in the interaction [10]. A busy server can always be better provisioned to meet the capacity requirements, e.g., by forming a server cluster. We shall discuss in Section IV-D how to deal with server capacity constraints in our proposed assignment algorithms if server capacities are limited.

The interaction between two clients c_i and c_j goes through their assigned servers. When c_i issues an operation, in order for c_j to see the effect of the operation, c_i first sends the operation to its assigned server $s_A(c_i)$. Then, $s_A(c_i)$ forwards the operation to c_j's assigned server $s_A(c_j)$ if they are different. Finally, $s_A(c_j)$ executes the operation and delivers the resultant state update to c_j. Thus, the interaction process involves the paths from c_i to $s_A(c_i)$, from $s_A(c_i)$ to $s_A(c_j)$, and from $s_A(c_j)$ to c_j. Similarly, when c_j issues an operation, the same three paths are involved in the interaction process for c_i to see the effect of the operation. Therefore, we refer to the concatenation of these three paths as the *interaction path* between c_i and c_j. The length of the interaction path, denoted by $d_A(c_i, c_j) = d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)$, represents the interaction time between c_i and c_j in assignment A. If c_i and c_j are assigned to the same server, the length of their interaction path is simply $d_A(c_i, c_j) = d(c_i, s_A(c_i)) + d(s_A(c_j), c_j)$. If c_i and c_j are the same client, the length of their interaction path $d_A(c_i, c_i) = 2d(c_i, s_A(c_i))$ is the round-trip time between c_i and its assigned server $s_A(c_i)$, and represents the interaction time for client c_i to see the effect of its own operation.

We use the average interaction time between all pairs of clients, i.e.,

$$\frac{2}{|C|(|C|+1)} \sum_{c_i, c_j \in C} d_A(c_i, c_j),^{1}$$

as a measure of the overall interactivity of the DIA. The average interaction time indicates how long it takes for the effect of an operation to be presented to a participant on an average level. Given the set of clients, the total number of client pairs is fixed. Thus, to minimize the average interaction time, it is equivalent to minimize the total length of interaction paths between all client pairs. Therefore, the client assignment problem is formulated as follows.

Definition 1: (Client Assignment Problem) Given a network $G = (V, E)$ where V contains a set of servers S and a set of clients C, and the length $d(u, v) > 0$ for each link $(u, v) \in E$, the objective of the client assignment problem is to find a client assignment A that minimizes the total length of interaction paths between all client pairs, i.e.,

$$\text{minimize} \quad D(A) = \sum_{c_i, c_j \in C} d_A(c_i, c_j).$$

Although the above formulation assumes that interaction exists between each pair of clients, our analysis and proposed algorithms can be easily generalized to handle the situation where each client only interacts with a portion of the other clients or to reflect different amounts of interaction between different pairs of clients.

III. NP-Completeness Results

We investigate the NP-completeness of the client assignment problem by assuming that messages are routed along shortest paths in the network. Under this assumption, $d(u, v)$ is the length of the shortest path between u and v, and the function $d(u, v)$ satisfies the triangle inequality. We start by studying the characteristics of an optimal client assignment, as shown in the following two theorems. Please refer to Appendices A and B in the supplementary file for the detailed proofs.

Theorem 1: If a server s_1 is located on the shortest path between a client c and another server s_2, then any assignment assigning c to s_2 cannot be an optimal assignment.

Theorem 2: If the shortest path between a client c_1 and a server s_1 and the shortest path between a client c_2 and a server s_2 have at least one common node u, then any assignment assigning c_1 and c_2 to s_1 and s_2 respectively cannot be an optimal assignment.

Making use of the above theorems, we show that the client assignment problem is NP-complete.

Theorem 3: The client assignment problem is NP-complete.

Proof: Consider a candidate solution for an instance of the client assignment problem in its decision version with a bound K. Since the length of the interaction path between each pair of clients can be computed in polynomial time, the computation of the total length of all interaction paths and its comparison with the bound K can be performed in polynomial time. Therefore, the client assignment problem is in NP.

We show that the client assignment problem is NP-complete by a polynomial reduction from the partition problem which is known to be NP-complete [11]. The partition problem is defined as follows: Given a finite set of positive integers B, does there exist a subset $B' \subseteq B$ such that $\sum_{b \in B'} b = \sum_{b \in B - B'} b$?

[1]We consider $d_A(c_i, c_j)$ as a function of an unordered pair of clients c_i and c_j (c_i can be the same as c_j). Given a set of clients C, there are a total of $|C|(|C|+1)/2$ unordered pairs of clients in the summation.

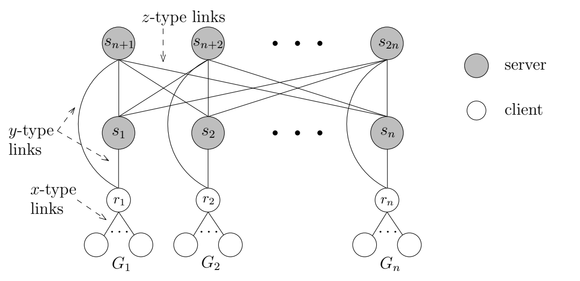


Fig. 1. Illustration of the instance Q of the client assignment problem.

Let P be an instance of the partition problem in which the set B has n elements: $B = \{b_1, b_2, \cdots, b_n\}$, and $\sum_{i=1}^{n} b_i = S$. We first construct a network with $2n$ servers $s_1, s_2, \cdots, s_{2n}$ and n groups of clients $G_1, G_2, \cdots, G_n$, as shown in Figure 1. The servers are divided into two sets $U_1 = \{s_1, s_2, \cdots, s_n\}$ and $U_2 = \{s_{n+1}, s_{n+2}, \cdots, s_{2n}\}$ with equal cardinality n. Each server in U_1 is connected to all servers in U_2 so that all the servers and inter-server links form a bipartite graph. Each client group G_i contains b_i clients. So, there are a total of $\sum_{i=1}^{n} b_i = S$ clients. In each client group, one client is designated as the center and the remaining clients are connected to the center to form a star graph. The center r_i of group G_i is connected to server s_i in U_1 and server s_{n+i} in U_2. An instance Q of the client assignment problem is then constructed on the network by setting the length of every link to 1, and the bound $K = \frac{11}{4}S^2 - (n-2)S - n - \sum_{i=1}^{n} b_i^2$. It is obvious that instance Q is constructed in time polynomial to the size of instance P. In the following, we show that there exists a subset B' such that $\sum_{b \in B'} b = \sum_{b \in B - B'} b$ for instance P if and only if there exists a client assignment such that its total length of interaction paths is at most K for instance Q.

In the network that we construct, servers s_i and s_{n+i} are on the shortest paths between the center client r_i and all other servers. According to Theorem 1, each center client r_i must be assigned to either s_i or s_{n+i} in an optimal assignment. In addition, each center client is on the shortest paths between all the other clients in the same group and all servers. According to Theorem 2, all clients in one group must be assigned to the same server in an optimal assignment. So, we only consider such assignments that assign all clients in group G_i to either server s_i in U_1 or server s_{n+i} in U_2.

Since the length of every link is 1, the total path length in an assignment can be calculated by adding up how many times each link appears in the interaction paths between all client pairs. We divide all links in the network into three types (see Figure 1): the x-type links between the center client and the remaining clients in the same group, the y-type links between the center clients and their directly connected servers, and the z-type links between servers. The occurrences of x-type and y-type links in interaction paths are independent of the client assignment. Each x-type link is associated with one client that is not a center client. The x-type link contributes twice to the interaction path between this client and itself and contributes once to the interaction path between this client and each of the other clients in the network. Since there are $S-n$ clients that are not the center clients, all x-type links contribute a total of $(S-n)(S+1)$ times. On the other hand, the interaction path between each pair of clients contains two y-type links. Thus, all y-type links contribute $S(S+1)$ times in total.

For z-type links, their contributions depend on how many clients are assigned to server sets U_1 and U_2. The interaction path between each pair of clients contains one z-type link if the pair of clients are assigned to servers in different server sets and contains two z-type links if the pair of clients are assigned to different servers in the same set. We denote by $a(s_i)$ the number of clients assigned to server s_i. Then, all z-type links contribute a total of

$$\sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) + 2 \cdot \sum_{\substack{s_i, s_j \in U_1 \\ s_i \neq s_j}} a(s_i)a(s_j) + 2 \cdot \sum_{\substack{s_i, s_j \in U_2 \\ s_i \neq s_j}} a(s_i)a(s_j)$$

times. Thus, the total length of interaction paths is given by

$$\begin{aligned}
&(S-n)(S+1) + S(S+1) + \sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) \\
&\quad + 2 \cdot \sum_{\substack{s_i, s_j \in U_1 \\ s_i \neq s_j}} a(s_i)a(s_j) + 2 \cdot \sum_{\substack{s_i, s_j \in U_2 \\ s_i \neq s_j}} a(s_i)a(s_j) \\
&= (S-n)(S+1) + S(S+1) + \sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) \\
&+ \Big(\sum_{s_i \in U_1} a(s_i)\Big)^2 - \sum_{s_i \in U_1} a(s_i)^2 + \Big(\sum_{s_i \in U_2} a(s_i)\Big)^2 - \sum_{s_i \in U_2} a(s_i)^2 \\
&= (S-n)(S+1) + S(S+1) + \Big(\sum_{s_i \in U_1 \cup U_2} a(s_i)\Big)^2 \\
&\quad - \sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) - \sum_{s_i \in U_1 \cup U_2} a(s_i)^2.
\end{aligned}$$

Since each client group G_i contains b_i clients, for servers $s_i \in U_1$ and $s_{n+i} \in U_2$, either $a(s_i) = b_i$ and $a(s_{n+i}) = 0$, or $a(s_i) = 0$ and $a(s_{n+i}) = b_i$. Therefore, for each possible assignment, $\sum_{s_i \in U_1} a(s_i)$ and $\sum_{s_i \in U_2} a(s_i)$ correspond to the sums of subsets B' and $B - B'$ respectively for a partitioning of B, and vice versa. It also follows that $\sum_{s_i \in U_1 \cup U_2} a(s_i) = \sum_{i=1}^{n} b_i = S$, $\sum_{s_i \in U_1 \cup U_2} a(s_i)^2 = \sum_{i=1}^{n} b_i^2$ and $\sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) \leq S^2/4$. Thus, the total length of interaction paths has a lower bound

$$\begin{aligned}
&(S-n)(S+1) + S(S+1) + S^2 - \frac{S^2}{4} - \sum_{i=1}^{n} b_i^2 \\
&= \frac{11}{4}S^2 - (n-2)S - n - \sum_{i=1}^{n} b_i^2 = K.
\end{aligned}$$

The lower bound K is achieved only when $\sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) = S^2/4$, i.e., $\sum_{s_i \in U_1} a(s_i) = \sum_{s_i \in U_2} a(s_i) = S/2$.

Hence, the theorem is proven. $\square$

IV. Heuristic Algorithms

In this section, we present and analyze three heuristic assignment algorithms. These heuristic algorithms do not assume any particular routing strategy and triangle inequality in the network. They compute client assignments based simply on the network latencies between different nodes, which can be obtained with existing tools like ping and King [12].

A. Nearest-Assignment

Nearest-Assignment is an intuitive algorithm that assigns each client to its nearest server. It is easy to implement and can be performed in a distributed manner because each client selects its server independently. The complexity for each client to compute its nearest server is $O(|S|)$. When assuming that the network latency satisfies the triangle inequality, *Nearest-Assignment* has an approximation ratio of 3. Please refer to Appendix C in the supplementary file for the detailed proof.

Theorem 4: The total length of interaction paths between all client pairs in *Nearest-Assignment* is within three times of that in an optimal assignment.

The approximation ratio of 3 is tight for *Nearest-Assignment*. Figure 2a presents an example where the ratio between the total interaction path lengths in *Nearest-Assignment* and in an optimal assignment can be arbitrarily close to 3. Here, $c_1, c_2, \cdots, c_n$ are clients and $s, s_1, s_2, \cdots, s_n$ are servers. The distance between each client and server s is a, and for each i, the distance between client c_i and server s_i is $a - \varepsilon$, where $0 < \varepsilon < a$. Thus, in *Nearest-Assignment*, each client c_i is assigned to server s_i. The length of the interaction path between each pair of different clients is $6a - 4\varepsilon$, and the length of the interaction path from each client to itself is $2a - 2\varepsilon$. Since there are n clients, the total length of all interaction paths in *Nearest-Assignment* A_N is given by

$$D(A_N) = \frac{n(n-1)}{2} \cdot (6a - 4\varepsilon) + n \cdot (2a - 2\varepsilon).$$

On the other hand, if the value of ε is very small, the optimal assignment A^* should assign all clients to server s, and has the total interaction path length of

$$D(A^*) = \frac{n(n-1)}{2} \cdot 2a + n \cdot 2a.$$

As the number of clients, n, goes towards infinity and ε approaches 0, the ratio between $D(A_N)$ and $D(A^*)$ approaches

$$\lim_{n\to\infty} \lim_{\varepsilon\to 0} \frac{D(A_N)}{D(A^*)} = \lim_{n\to\infty} \frac{n(n-1)\cdot 3a + n \cdot 2a}{n(n-1)\cdot a + n \cdot 2a} = 3.$$

This shows that the ratio between $D(A_N)$ and $D(A^*)$ can be made arbitrarily close to 3.

In the absence of the triangle inequality [13], *Nearest-Assignment* has unbounded approximation ratio. Figure 2b shows an example network containing two clients c_1, c_2 and four servers s_1, s_2, s_3, s_4. The total interaction path length in *Nearest-Assignment* is $2 + 2 + (l + 2) = l + 6$, whereas the total interaction path length in the optimal assignment is $4+4+6 = 14$. Thus, the ratio between the total interaction path lengths in *Nearest-Assignment* and in the optimal assignment can be made arbitrary large as l goes towards infinity.

B. Modify-Assignment

Modify-Assignment is a local search heuristic. It starts with an existing assignment and continues to modify the assignment in a greedy manner for reducing the total length of interaction paths. To perform an assignment modification, *Modify-Assignment* examines all clients to find whether reassigning any client to a different server would reduce the total length of interaction paths. If there exist such reassignment options, the algorithm then selects the option that produces the maximum reduction and reassigns the client. *Modify-Assignment* repeats the above process until the total length of interaction paths cannot be further reduced by reassigning any client. Since the assignment modifications can only improve interactivity, the resultant assignment cannot be worse than the initial assignment.

When a client c is reassigned to a different server, only the interaction paths involving c would change. Let A be the current client assignment. Then, the total length of all interaction paths involving a client c is given by

$$\sum_{c_i \in C} \Big(d(c, s_A(c)) + d(s_A(c), s_A(c_i)) + d(s_A(c_i), c_i) \Big)$$
$$= (|C|{+}1) d(c, s_A(c)) {+} \sum_{c_i \neq c} d(s_A(c), s_A(c_i)) {+} \sum_{c_i \neq c} d(s_A(c_i), c_i),$$

where $|C|$ is the number of clients. If client c is reassigned to a different server s, the total length of all interaction paths involving c becomes

$$(|C|{+}1) d(c, s) + \sum_{c_i \neq c} d(s, s_A(c_i)) + \sum_{c_i \neq c} d(s_A(c_i), c_i).$$

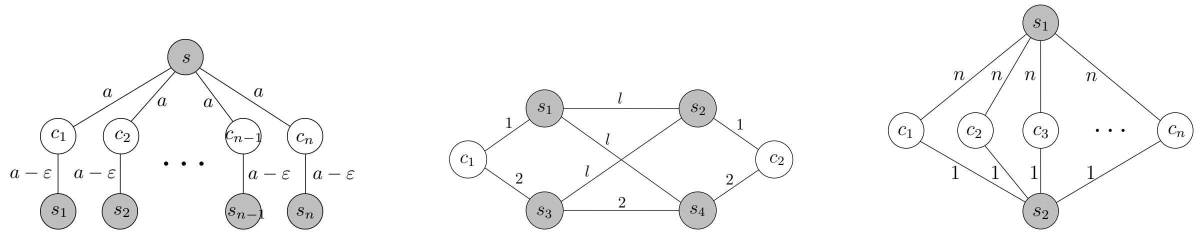
Therefore, the reduction in interaction path length is

$$(|C| + 1) d(c, s_A(c)) + \sum_{c_i \neq c} d(s_A(c), s_A(c_i))$$
$$- \Big((|C| + 1) d(c, s) + \sum_{c_i \neq c} d(s, s_A(c_i)) \Big). \qquad (1)$$

If we denote by $a(s)$ the number of clients that are assigned to server s in assignment A, $\sum_{c_i \neq c} d(s_A(c), s_A(c_i))$ can be rewritten as $\sum_{s_j \in S} a(s_j) d(s_A(c), s_j) - d(s_A(c), s_A(c)) = \sum_{s_j \in S} a(s_j) d(s_A(c), s_j)$. Similarly, $\sum_{c_i \neq c} d(s, s_A(c_i))$ can be rewritten as $\sum_{s_j \in S} a(s_j) d(s, s_j) - d(s, s_A(c))$. Thus, the reduction (1) can be rewritten as

$$(|C|{+}1) d(c, s_A(c)) + \sum_{s_j \in S} a(s_j) d(s_A(c), s_j)$$
$$- \Big((|C|{+}1) d(c, s) + \sum_{s_j \in S} a(s_j) d(s, s_j) - d(s, s_A(c)) \Big). \quad (2)$$

To efficiently calculate the two summations in (2), the values of $\sum_{s_j \in S} a(s_j) d(s, s_j)$ for all servers s can be stored in an array L and incrementally updated after each assignment modification. The pseudocode of *Modify-Assignment* is presented in Algorithm 1. The calculation of array L for the initial assignment (lines 2 to 3) has a complexity of $O(|S|^2)$, where $|S|$ is the number of servers. The computational complexity of each assignment modification (lines 5 to 15) is $O(|S||C|)$. Thus, given an initial assignment, the total computational complexity of *Modify-Assignment* is $O(|S|^2 + n|S||C|)$, where n is the number of assignment modifications performed before the algorithm terminates. Our experimental results in Section



(a) The approximation ratio 3 of *Nearest-Assignment* is tight.

(b) Unbounded approximation ratio when the network latency does not satisfy the triangle inequality.

(c) The approximation ratio $|C|$ of *Modify-Assignment* and *Distributed-Modify-Assignment* is tight.

Fig. 2. Examples illustrating that the analysis of approximation ratios is tight.

Algorithm 1: The *Modify-Assignment* algorithm

```
1  for each server $s_j$, let $a(s_j)$ be the number of clients assigned
   to $s_j$ in the initial assignment;
2  foreach $s \in S$ do
3      $L[s] \leftarrow \sum_{s_j \in S} a(s_j) \cdot d(s, s_j)$;
4  repeat
5      $\Delta^* \leftarrow -\infty$;
6      foreach $s \in S$ do
7          foreach $c \in C$ do
8              $\Delta \leftarrow (|C|+1)d(c, s_A(c)) + L[s_A(c)] - \big((|C|+1)d(c,s) + L[s] - d(s, s_A(c))\big)$;
9              if $\Delta > 0$ & $\Delta > \Delta^*$ then
10                 $\Delta^* \leftarrow \Delta$;
11                 $c^* \leftarrow c$;   $s^* \leftarrow s$;
12     if $\Delta^* > 0$ then
13         foreach $s \in S$ do
14             $L[s] \leftarrow L[s] - d(s, s_A(c^*)) + d(s, s^*)$;
15         set $s_A(c^*) = s^*$;
16 until $\Delta^* \leq 0$ ;
```

V would show that n is usually less than $|C|$.

In the absence of the triangle inequality, the example in Figure 2b also shows that *Modify-Assignment* has unbounded approximation ratio (when it starts with c_1 and c_2 initially assigned to s_1 and s_2 respectively). If the network latency satisfies the triangle inequality, we can show that *Modify-Assignment* has an approximation ratio of $|C|$.

Theorem 5: Irrespective of the initial client assignment, the total length of all interaction paths in *Modify-Assignment* is within $|C|$ times of that in an optimal assignment.

Proof: Suppose there are n clients $c_1, c_2, \cdots, c_n$ in the network. We denote by m_i the assigned server of client c_i in *Modify-Assignment* and denote by o_i the assigned server of c_i in an optimal assignment.

When the *Modify-Assignment* algorithm terminates, changing the assigned server of any client c_i to o_i cannot reduce the total length of all interaction paths involving c_i. Therefore, for each client c_i,

$$2d(c_i, m_i) + \sum_{j \neq i} \Big(d(c_i, m_i) + d(m_i, m_j) + d(m_j, c_j)\Big) \leq 2d(c_i, o_i) + \sum_{j \neq i} \Big(d(c_i, o_i) + d(o_i, m_j) + d(m_j, c_j)\Big),$$

which implies that

$$(n{+}1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j) \leq (n{+}1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, m_j). \tag{3}$$

By the triangle inequality, we have

$$d(o_i, m_j) \leq d(o_i, c_i) + d(c_i, m_i) + d(m_i, m_j), \tag{4}$$

$$d(o_i, m_j) \leq d(o_i, o_j) + d(o_j, c_j) + d(c_j, m_j). \tag{5}$$

Combining inequalities (3) and (4), we obtain

$$\begin{aligned} &(n+1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j) \\ &\leq (n{+}1)d(c_i, o_i) + \sum_{j \neq i} \Big(d(o_i, c_i) + d(c_i, m_i) + d(m_i, m_j)\Big) \\ &= (n+1)d(c_i, o_i) + (n-1)d(o_i, c_i) \\ &\quad + (n-1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j), \end{aligned}$$

which indicates that $d(c_i, m_i) \leq n \cdot d(c_i, o_i)$, and hence

$$n \cdot \sum_{i=1}^{n} d(c_i, m_i) \leq n^2 \cdot \sum_{i=1}^{n} d(c_i, o_i). \tag{6}$$

Combining inequalities (3) and (5), we have

$$\begin{aligned} &(n+1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j) \\ &\leq (n{+}1)d(c_i, o_i) + \sum_{j \neq i} \Big(d(o_i, o_j) + d(o_j, c_j) + d(c_j, m_j)\Big). \end{aligned}$$

By adding up the above inequality for all clients c_is, we obtain

$$\begin{aligned} &(n+1)\sum_{i=1}^{n} d(c_i, m_i) + \sum_{i=1}^{n}\sum_{j \neq i} d(m_i, m_j) \\ &\leq (n{+}1)\sum_{i=1}^{n} d(c_i, o_i) + \sum_{i=1}^{n}\sum_{j \neq i} \Big(d(o_i, o_j) + d(o_j, c_j) + d(c_j, m_j)\Big) \\ &= (n+1)\sum_{i=1}^{n} d(c_i, o_i) + \sum_{i=1}^{n}\sum_{j \neq i} d(o_i, o_j) \\ &\quad + (n-1)\sum_{i=1}^{n} d(o_i, c_i) + (n-1)\sum_{i=1}^{n} d(c_i, m_i). \end{aligned}$$

It follows that

$$\sum_{i=1}^{n} d(c_i, m_i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i} d(m_i, m_j) \leq n\sum_{i=1}^{n} d(c_i, o_i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i} d(o_i, o_j). \quad (7)$$

Finally, adding up (6) and (7), we have

$$\begin{aligned}(n+1)\sum_{i=1}^{n} d(c_i, m_i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i} d(m_i, m_j) &\leq n(n+1)\sum_{i=1}^{n} d(c_i, o_i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i} d(o_i, o_j)\\ &\leq n\Big((n+1)\sum_{i=1}^{n} d(c_i, o_i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i} d(o_i, o_j)\Big). \quad (8)\end{aligned}$$

Note that the left side of inequality (8) is exactly the total length of all interaction paths in *Modify-Assignment*, and the right side of inequality (8) is exactly n times of the total length of all interaction paths in the optimal assignment. Therefore, *Modify-Assignment* has an approximation ratio of $n = |C|$.

Hence the theorem is proven. □

The approximation ratio of $|C|$ is tight if *Modify-Assignment* starts with an arbitrary initial assignment. Figure 2c shows an example network containing n clients $c_1, c_2, \cdots, c_n$ and two servers s_1, s_2. The distance between each client and server s_1 is n, and the distance between each client and server s_2 is 1. Suppose that in the initial assignment, all clients are assigned to server s_1. Then, the total length of all interaction paths involving any client c_i is $2n^2$. If client c_i is reassigned to server s_2, the total length of all interaction paths involving c remains $2 + (2n+2)(n-1) = 2n^2$. Thus, *Modify-Assignment* would terminate without reassigning any client. On the other hand, the optimal assignment is to assign all clients to server s_2 so that the interaction path length between any two clients is 2, which is $1/n$ of that in *Modify-Assignment*. Therefore, the total interaction path length in *Modify-Assignment* is $n = |C|$ times of that in the optimal assignment.

If *Modify-Assignment* takes *Nearest-Assignment* as the initial assignment, the resultant assignment cannot be worse than *Nearest-Assignment*. Our experimental results in Section V would show that *Modify-Assignment* normally improves interactivity over *Nearest-Assignment* significantly.

C. Distributed-Modify-Assignment

Distributed-Modify-Assignment is a distributed version of *Modify-Assignment*, in which assignment modifications are performed in a distributed manner. Starting from an initial assignment, each client individually changes its assigned server to the server that decreases the total interaction path length most. The algorithm terminates if no client can reduce the total interaction path length by changing its assigned server.

Similar to *Modify-Assignment*, a client c only needs to consider the interaction paths involving itself in performing an assignment modification. As derived in Section IV-B, the total length of these interaction paths is given by $(|C|+1)d(c,s) + \sum_{s_j\in S} a(s_j)d(s,s_j) - d(s,s_A(c)) + \sum_{c_i\neq c} d(s_A(c_i), c_i)$, if c changes its assigned server to s. Note that the last summation is independent of s and can be omitted for comparison purposes. Thus, this expression becomes

$$(|C|+1)d(c,s) + \sum_{s_j\in S} a(s_j)d(s,s_j) - d(s,s_A(c)). \quad (9)$$

The calculation needs the information of the latency $d(c,s)$ between c and s, the latencies between s and other servers, and the number of clients assigned to each server s_j (i.e., $a(s_j)$). To this end, each server maintains the latencies from itself to the other servers, and one master server keeps track of how many clients are assigned to each server. Note that the last two terms of (9), which we shall denote by $H(s) = \sum_{s_j\in S} a(s_j)d(s,s_j) - d(s,s_A(c))$, are independent of the client-to-server latency. To perform an assignment modification, client c's server $s_A(c)$ queries the master server and calculates $H(s)$ for all the other servers $s \neq s_A(c)$. If the current value of (9), i.e., $(|C|+1)d(c,s_A(c)) + H(s_A(c))$, is not greater than $H(s)$, changing c's server to s cannot reduce the interaction path length. Thus, $s_A(c)$ informs c about the list of servers whose $H(s)$ is less than $(|C|+1)d(c,s_A(c)) + H(s_A(c))$. Then, c contacts each server s in the list to measure the latency to the server and calculates the value of (9) for this server. Based on the calculation results, c decides locally whether and how to change its assigned server. The computational complexity of an assignment modification is $O(|S|^2)$.

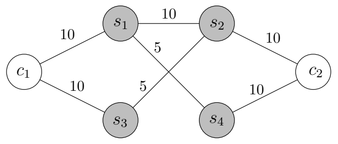


Fig. 3. An example in which changing two clients' assigned servers simultaneously increases the total interaction path length.

One issue with *Distributed-Modify-Assignment* is that, if two or more clients change their assigned servers simultaneously, the total interaction path length is not guaranteed to decrease because the calculation of each client is based on the assumption that the assigned servers of other clients remain unchanged. Figure 3 gives an example, where the network contains clients c_1, c_2 and servers s_1, s_2, s_3, s_4, and the number on each link represents the length of the link. Suppose the initial assignment assigns c_1 to s_1 and c_2 to s_2. The lengths of interaction paths between client pairs (c_1, c_2), (c_1, c_1) and (c_2, c_2) are 30, 20 and 20 respectively. So, the total path length is 70. When client c_1 tries to change its assigned server, it would choose s_3 since the length of the interaction path between c_1 and c_2 would be reduced to 25 while the lengths of interaction paths between client pairs (c_1, c_1) and (c_2, c_2) stay the same, assuming that c_2 keeps its assigned server unchanged. Similarly, client c_2 would choose to change its assigned server to s_4. However, if the two clients both change their assigned servers at the same time, the path length between c_1 and c_2 would be increased to 40 which is even longer than that in the initial assignment. This issue can be resolved by a concurrency control mechanism to ensure that assignment modification proceeds one client at a time. To do so, a token can be maintained by the servers and circulated among clients. Only the client obtaining the token is allowed to perform an assignment modification. After the assignment modification, the client returns the token to its server which passes the token onto another client.

Distributed-Modify-Assignment has the same approximation ratio as *Modify-Assignment*. The proof of Theorem 5 and the examples of Figures 2b and 2c also apply to *Distributed-Modify-Assignment*.

D. Dealing with Limited Server Capacities

So far, we have not assumed any limitation on the capacity of each server. In practice, servers have finite capacities. If the number of clients assigned to a server exceeds its capacity, the processing delay at the server can increase significantly [14]. Simple steps can be taken in our proposed algorithms to ensure that the capacity of each server is not exceeded. In *Nearest-Assignment*, if the nearest server of a client is saturated, the client turns to the next nearest server and keeps doing so until it finds a server that can accommodate more clients. For *Modify-Assignment* and *Distributed-Modify-Assignment*, only the servers not yet saturated are considered when performing assignment modifications. We evaluate both "uncapacitated" and "capacitated" assignment algorithms in the next section.

The approximation ratios of the "uncapacitated" algorithms do not apply directly to the "capacitated" algorithms. Figure 4 gives an example when the "capacitated" *Nearest-Assignment* is more than three times worse than the optimal assignment. Each server in this network can accommodate one client only. Suppose that in *Nearest-Assignment*, client c_1 is first assigned to server s_2. Then, each remaining client c_i ($i = 2, 3, 4$) can only be assigned to server s_{i+1}. The total interaction path length under this assignment is 1190. The optimal assignment should assign each c_i ($i = 1, 2, 3, 4$) to s_i, which produces a total interaction path length of 322. Thus, the ratio between the results of the two assignments is 3.7. We leave the theoretical analysis of the approximation ratios for the "capacitated" assignment algorithms to the future work.

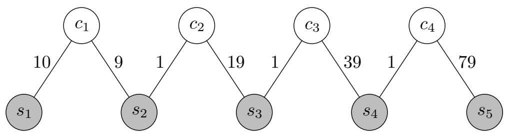


Fig. 4. The approximation ratio of the "capacitated" *Nearest-Assignment* is larger than 3.

V. Experimental Evaluation

A. Experimental Setup

To evaluate the proposed assignment algorithms, we have conducted extensive simulation experiments by making use of the Meridian data set [15]. This is a real network latency data set that contains pair-wise latency measurements between 2500 nodes in the Internet using the King technique [12]. The measurements for some node pairs are not available in the data set. On discarding the nodes involved in unavailable measurements, the network simulated in our experiments is represented by a complete pair-wise latency matrix between 1796 nodes. A client is assumed to be located at each node and a certain number of servers are assumed to be placed at selected nodes in the network. The proposed client assignment algorithms are evaluated with different parameter settings of server number and server capacity. In our experiments, we choose *Nearest-Assignment* as the initial assignment for both *Modify-Assignment* and *Distributed-Modify-Assignment*. In the default setting, no limitation is assumed on the server capacity.

We simulate three types of server placements: random, k-median and k-center. The k-median placement aims to place a given number of k servers in the network to minimize the total distance from the clients to their closest servers. The k-center placement targets at placing k servers in the network to minimize the maximum distance between a client and its closest server. The k-median and k-center problems are widely used to model server placement in the Internet [10], [16]. Both problems are NP-hard [11]. We adopt a greedy k-median heuristic [16] and a greedy k-center heuristic [10] for placing servers in our experiments. For random server placement, under each parameter setting, we perform $1,000$ simulation runs using $1,000$ different sets of servers placed at random. Unless stated otherwise, the average performance of these simulation runs together with the error bars of standard deviation is plotted for performance comparison.

To quantify the relative performance difference, the interactivity performance of different algorithms is normalized with respect to a common theoretical lower bound, which is derived as follows. For any client assignment A, the interaction path between two clients u, v has a length of

$$\begin{aligned} d_A(u,v) &= d(u,s_A(u)) + d(s_A(u),s_A(v)) + d(s_A(v),v) \\ &\geq \min_{s_i,s_j\in S}\{d(u,s_i)+d(s_i,s_j)+d(s_j,v)\}. \end{aligned}$$

By adding up the above inequalities for all client pairs, we get the following lower bound on the total interaction path length:

$$\sum_{u,v\in C}\big(\min_{s_i,s_j\in S}\{d(u,s_i)+d(s_i,s_j)+d(s_j,v)\}\big).$$

Note that this lower bound is a super bound that may not be achievable by any client assignment. This is because the above bound calculation does not enforce the constraint that each client is assigned to only one server through which it interacts with all the other clients. The total length of interaction paths produced by each algorithm normalized by the above bound shall be called the *normalized interactivity*.

We also include in the comparison an existing heuristic *MAXFLO* proposed in [17] for a similar problem called the fixed-hub single allocation problem. We adapt this heuristic to our client assignment problem. The first step of the heuristic is similar to the above calculation of the lower bound, in which we find the assigned servers for each pair of clients that produce the shortest interaction path between the two clients. In the second step, each client is assigned to the server to which it is assigned for the most times in the first step. We

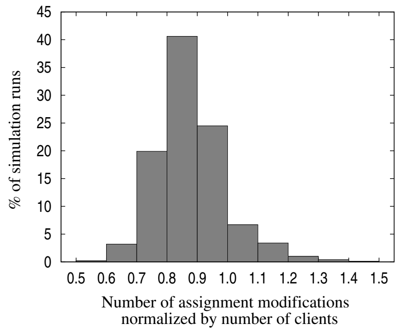


Fig. 5. Distribution of the number of assignment modifications in *Modify-Assignment* for 80 servers under random server placement.

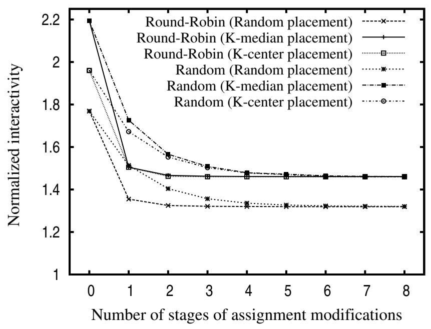


Fig. 6. Performance of *Distributed-Modify-Assignment* after different numbers of stages of assignment modifications for 80 servers.

compare *MAXFLO* with our "uncapacitated" algorithms.

B. Number of Assignment Modifications

We first investigate the number of assignment modifications performed by *Modify-Assignment* and *Distributed-Modify-Assignment*. Figure 5 shows the distribution of the numbers of assignment modifications required for *Modify-Assignment* to terminate under $1,000$ different sets of 80 randomly placed servers. The number of modifications is normally less than the number of clients (i.e., each client performs less than one modification on average). The numbers of assignment modifications under the k-median and k-center placements are measured to be $1,620$ and $1,289$ respectively, which are also less than the number of clients. Similar observations are made for other server numbers as well.

Unlike *Modify-Assignment* which performs assignment modifications based on global knowledge, *Distributed-Modify-Assignment* makes modification decisions locally at individual clients. Therefore, *Distributed-Modify-Assignment* generally needs more assignment modifications than *Modify-Assignment*. We count an attempt made by a client to change its assigned server as one assignment modification, even if the client decides not to change its assigned server after calculation. n successive assignment modifications are referred to as *one stage* if there are n clients. We study the convergence speed of *Distributed-Modify-Assignment* by monitoring its interactivity performance after each stage of assignment modifications. Two schemes for circulating the token among clients are tested: random and round-robin. Random circulation passes the token

TABLE I
COMPUTATION TIMES

Algorithm	CPU Time
Nearest-Assignment	0.16s
Modify-Assignment	18.75s
Distributed-Modify-Assignment (Random)	2.13s
Distributed-Modify-Assignment (Round-Robin)	3.52s
MAXFLO	27.40s

to a randomly selected client after each assignment modification, whereas round-robin circulation passes the token to each client exactly once in a stage of assignment modifications. Figure 6 presents the results for 80 servers under different server placements. As can be seen, the normalized interactivity is continuously improved with increasing number of stages and the algorithm converges quickly in just a few stages. Random and round-robin circulations achieve more than 90% and 99% of the improvements over the initial assignment respectively within three stages (i.e., after each client has performed three assignment modifications on average). Similar trends are also observed for other server numbers. Thus, in the following experiments, we shall execute the *Distributed-Modify-Assignment* algorithm for only three stages of assignment modifications for both token passing schemes.

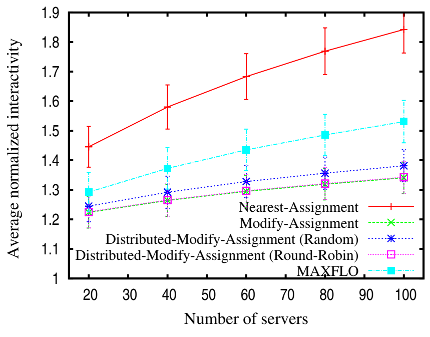
C. Performance Comparison for Different Algorithms

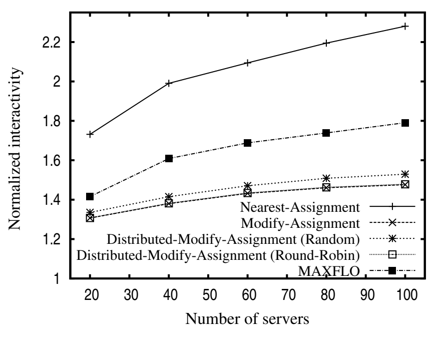
Now, we compare the performance of different client assignment algorithms. Figure 7 shows the normalized interactivity produced by the algorithms for different server numbers and server placements. It can be seen that *Modify-Assignment* produces the best results, and *Distributed-Modify-Assignment* performs very close to its centralized counterpart. The interactivities resulting from *Modify-Assignment* and *Distributed-Modify-Assignment* are normally within 40% to 50% of the lower bound, which means that they are within at most the same percentages of the optimum. Round-robin circulation can improve the performance of *Distributed-Modify-Assignment* over random circulation, but the improvement is limited (within 5%). Note that additional information needs to be maintained by the server to ensure that each client obtains the token exactly once in a stage of assignment modifications. Thus, we shall use random circulation for passing the token in the following experiments. *Nearest-Assignment* produces the worst interactivity among all the algorithms. This implies that reducing the client-to-server latency alone is not very effective in improving interactivity. *MAXFLO* outperforms *Nearest-Assignment*, but its performance is still far worse than our modification-based algorithms.

Table I reports the average computation times of the algorithms under 80 randomly placed servers in our simulation performed on a PC with a Pentium Xeon 3.2 GHz CPU and 6GB RAM. It can be seen that *Distributed-Modify-Assignment* has much shorter computation time than *Modify-Assignment* and *MAXFLO*.

D. Impact of Server Capacity

We also evaluate the "capacitated" version of the assignment algorithms. We refer to the maximum number of clients that


(a) Random placement


(b) k-median placement

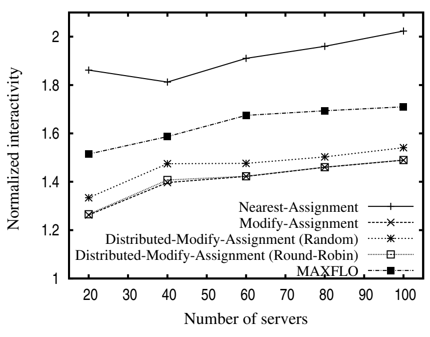
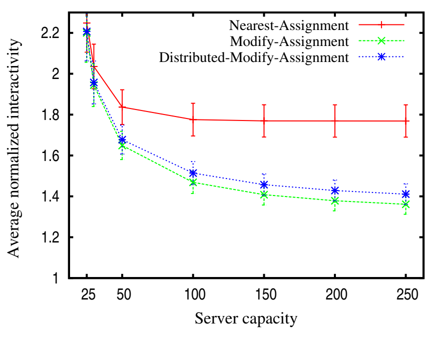
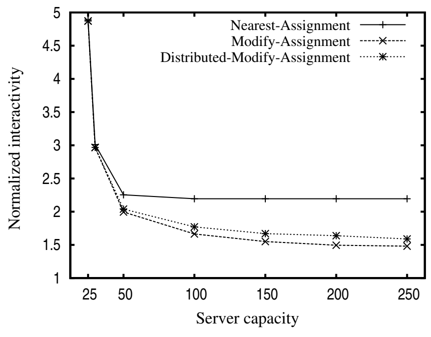

(c) k-center placement

Fig. 7. Performance of client assignment algorithms for different numbers of servers.


(a) Random placement


(b) k-median placement

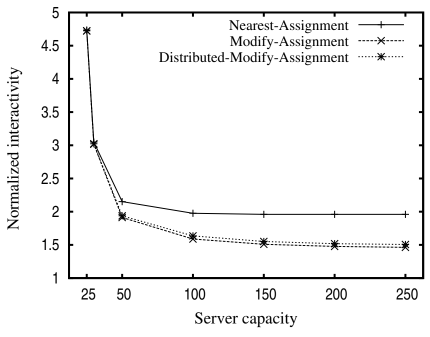

(c) k-center placement

Fig. 8. Performance of client assignment algorithms for different server capacities.

can be assigned to each server as the server capacity. Figure 8 shows the normalized interactivity as a function of the server capacity when there are 80 servers placed in the network. Note that the theoretical lower bound does not change with server capacity as it assumes unlimited server capacity. As can be seen, the interactivity resulting from all algorithms generally becomes worse with decreasing server capacity. This is because lower server capacities make the algorithms unable to assign the clients to the most desirable servers. In general, the relative performance of all algorithms remains unchanged over different server capacities. Only when the server capacity is severely limited do all algorithms produce similar interactivity, which is much worse than that when servers have abundant capacities. Similar results are observed for other server numbers.

E. Dynamic Scenarios

The above experiments have shown that *Modify-Assignment* results in the best interactivity among all algorithms in static scenarios where the participating clients and network latencies are fixed. However, *Modify-Assignment* is a centralized algorithm that requires global knowledge about the network. On the other hand, *Distributed-Modify-Assignment*, which performs close to *Modify-Assignment*, is a distributed algorithm that is more suitable for incrementally adapting the client assignment to dynamics in client participation and network latencies. In this section, we evaluate the adaptivity of *Distributed-Modify-Assignment* in dynamic scenarios. For

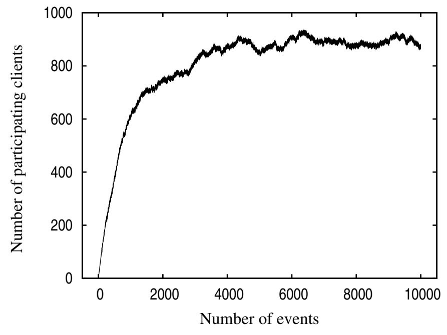


Fig. 9. The number of participating clients over the event sequence.

comparison purposes, we also include *Nearest-Assignment* in the experiments.

1) Adaptivity to Client Joining/Leaving: We first consider the scenario in which the set of participating clients changes over time due to joins and leaves. We simulate it by a sequence of randomly generated client join and leave events. To model clients with different join and leave patterns, each client is assigned two random weights w_j and w_l ($0 < w_j, w_l < 1$). If a client is currently participating, the probability of generating a leave event for the client is proportional to its w_j; if a client is not participating, the probability of generating a join event for the client is proportional to its w_l. As a result, clients with higher w_j and w_l weights would join and leave the system more frequently than clients with lower w_j and w_l

weights. Clients with high w_j but low w_l tend to participate in the system for longer time periods than clients with low w_j but high w_l. The simulation run starts with no client participating in the system and executes a sequence of $10,000$ join and leave events. Figure 9 shows the number of clients participating in the system over the event sequence. As can be seen, after a warm-up period of about $5,000$ events, the number of participating clients remains stable at about 900, which is about half the total number of clients in the network because the weights of all clients are randomly selected.

To adapt the client assignment to dynamic client participation, *Nearest-Assignment* simply assigns all joining clients to their nearest servers. For *Distributed-Modify-Assignment*, a certain number of assignment modifications are performed after each join and leave event to refine the assignment. In the case of a join event, the modifications start with the joining client. Note that all assignment modifications are made based on the existing client assignment. Figure 10 shows the interactivity performance of different algorithms when there are 80 servers placed in the network.[2] As can be seen, while aggressively performing three stages of assignment modifications at every event produces the best interactivity among all algorithms, performing as few as just one assignment modification at every event results in almost the same performance. This implies that *Distributed-Modify-Assignment* is able to adapt the client assignment quickly to dynamic client participation. Performing only a few assignment modifications per event is sufficient to maintain decent interactivity performance.

We also study the communication overhead of *Distributed-Modify-Assignment* by counting the number of servers that are contacted by the client in each assignment modification. On average, the numbers of servers contacted in an assignment modification are 38, 26 and 25.5 for random, k-median and k-center server placements respectively, which are less than $1/3$ to half of all the servers in the network.

2) Adaptivity to Dynamics in Network Conditions: The network latency between nodes may change over time due to dynamics in network conditions. In this section, we evaluate the adaptivity of the algorithms to the variance in network latency. Since the Meridian data set does not contain latency data at different times, we use the PlanetLab-All-Pairs-Pings data set [18] which includes periodic pair-wise latency measurements between PlanetLab nodes. The measurements were performed approximately once every 15 minutes. We use the data collected over a one day period (on May 9, 2005) that contains 95 measurements. There are 188 PlanetLab nodes whose pair-wise latency measurements are consistently available in all of these 95 measurements. Thus, we simulate a network consisting of these 188 nodes. Similar to the previous experiments, a client is assumed to be located at each node and a total number of 15 servers are assumed to be placed at selected nodes in the network.

To deal with dynamic network latency, *Nearest-Assignment* simply lets each client recalculate its nearest server after every measurement. For *Distributed-Modify-Assignment*, three stages of assignment modifications are performed after every measurement to refine the existing client assignment.

Figure 11 shows the interactivity performance of the two algorithms over the one day period. Here, the k-median and k-center server placements are computed based on the latency data of the first measurement. The theoretical lower bound on the total length of interaction paths is recalculated after each measurement for computing the normalized interactivity. It can be seen from Figure 11 that *Distributed-Modify-Assignment* almost always outperforms *Nearest-Assignment*. The normalized interactivity of *Nearest-Assignment* fluctuates widely. This is because *Nearest-Assignment* considers the client-to-server latencies only. Thus, its relative performance would become worse when the inter-server latencies between the nearest servers of the clients increase. In contract, *Distributed-Modify-Assignment* can adjust to changes in both the client-to-server latencies and inter-server latencies. As a result, its normalized interactivity remains quite stable over time, implying that it adapts well to the variance in network latency. In *Distributed-Modify-Assignment*, similar to the results in the previous scenario, only less than $1/3$ to half of the servers are typically contacted in an assignment modification. Note that the client that performs an assignment modification is not necessarily reassigned. Our results show that only less than 7% clients change their assigned servers after each measurement although three stages of assignment modifications are performed.

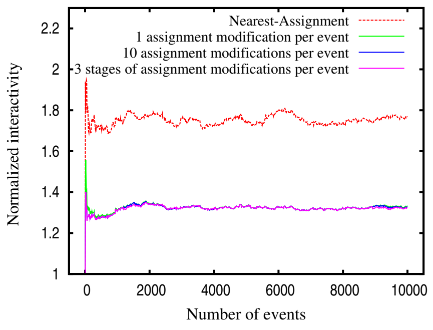
VI. Optimal Assignment for Tree Topologies

Overlay tree topologies are widely used to facilitate large-scale group communication in distributed applications [19]. For example, a shared spanning tree topology can be built for efficiently supporting multicast communication among distributed servers and from the servers to their clients [20]. While the client assignment problem for DIAs is NP-complete for general topologies, there do exist polynomial optimal solutions for tree topologies. In this section, we develop a dynamic programming algorithm to compute the optimal client assignment in tree topologies. For simplicity, we assume that there is a client located at each node in the network. Extending the solution to the case where clients are located at only a subset of nodes is straightforward.

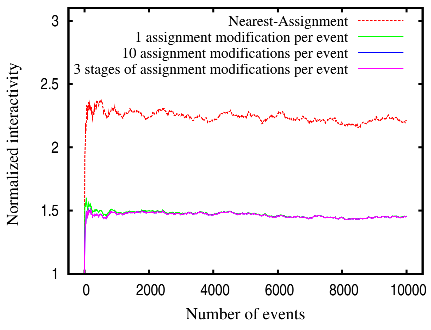
A. Generalized Problem Definitions

In a tree topology, each internal node u and its descendant nodes form a subtree rooted at u. As shown in Figure 12a, we denote the subtree rooted at node u by T_u and denote the set of links incident upon the nodes in subtree T_u by E_u. Note that E_u includes all links whose both ends are located inside T_u as well as the link connecting u to its parent $p(u)$ if u is not the root of the entire tree. We denote by $\mathcal{C}(E_u)$ the contribution of the links E_u to the total length of all interaction paths between clients. It is obvious that $\mathcal{C}(E_u)$ is affected only by the assignment of the clients inside T_u and how many clients outside T_u are assigned to servers inside T_u. Thus, to develop a dynamic programming algorithm, we consider a generalized problem of how to assign the clients inside T_u to minimize

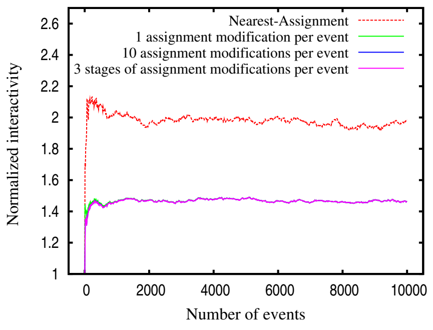
[2]The results for random server placement in Figure 10a are those for a sample set of randomly placed servers. We have performed simulations with many different sets of randomly placed servers and observed similar performance trends.



(a) Random placement

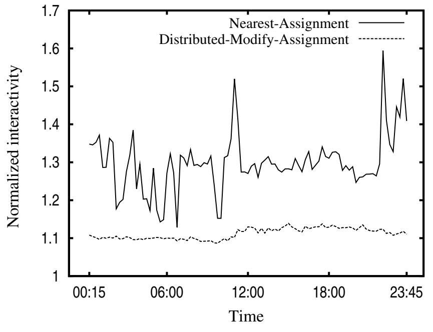


(b) k-median placement

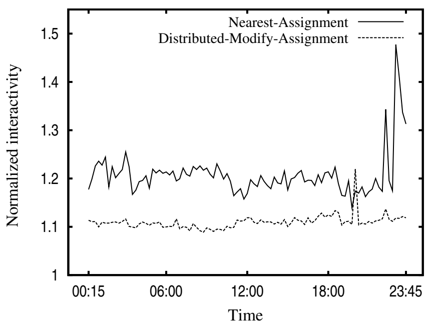


(c) k-center placement

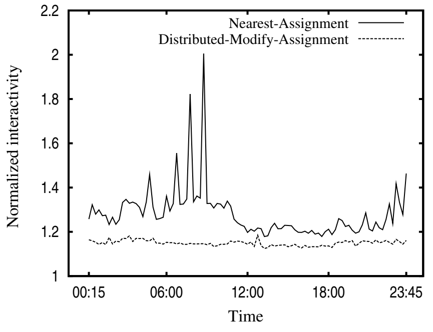
Fig. 10. Performance of client assignment algorithms with dynamic client participation.



(a) Random placement



(b) k-median placement



(c) k-center placement

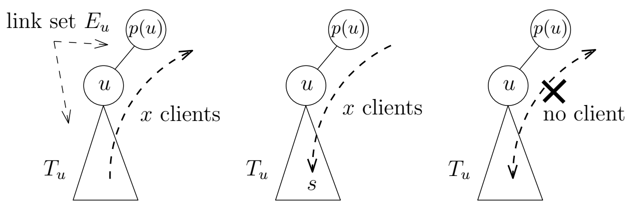
Fig. 11. Performance of client assignment algorithms with dynamic network latency.

the contribution $\mathcal{C}(E_u)$ given that a certain number of clients outside T_u are assigned to servers inside T_u.

Note that the path between any two nodes in a tree is unique and is the shortest path connecting these two nodes. Thus, both Theorems 1 and 2 are applicable to client assignment in the tree topology. According to Theorem 2, in an optimal assignment, if any client is assigned to a server through a path that involves a node u, then no client would be assigned to other servers through paths that involve u. As a result, the clients outside subtree T_u can be assigned to servers inside T_u only if no client inside T_u is assigned to any server outside T_u. In addition, if multiple clients outside T_u are assigned to servers inside T_u, they must be all assigned to the same server inside T_u. Therefore, we classify the generalized problem into three types defined as follows. To facilitate presentation, we refer to an assignment of the clients inside subtree T_u as a T_u-client assignment. We shall also use $|T_u|$ to refer to the number of nodes in the subtree T_u.

Definition 2: Assume that a client is located at each node in a tree network T and there are a set of servers S in T. Let u be a node in T.

1) Suppose that no client outside T_u is assigned to any server inside T_u. Given a number x ($1 \le x \le |T_u|$), among all T_u-client assignments that assign x clients inside T_u to servers outside T_u, the problem of finding a T_u-client assignment that minimizes $\mathcal{C}(E_u)$ is referred to as the (u, x)-*problem* (Figure 12a).
2) Suppose that no client inside T_u is assigned to any server outside T_u. Given a server s inside T_u and a number x ($1 \le x \le |T| - |T_u|$), assuming that x clients outside T_u are assigned to server s, the problem of finding a T_u-client assignment that minimizes $\mathcal{C}(E_u)$ is referred to as the (u, x, s)-*problem* (Figure 12b).
3) Suppose that no client outside T_u is assigned to any server inside T_u, and no client inside T_u is assigned to any server outside T_u. The problem of finding a T_u-client assignment that minimizes $\mathcal{C}(E_u)$ is referred to as the $(u, 0)$-*problem* (Figure 12c).



(a) (u, x)-problem (b) (u, x, s)-problem (c) $(u, 0)$-problem

Fig. 12. (Definition 2) Generalized problems.

Suppose r is the root of the entire tree T. Then, no client is outside T_r, and E_r is the set of all links in T. So, the $(r, 0)$-problem is the original client assignment problem in tree T.

B. Recurrences for Dynamic Programming

We first show that for each internal node u in tree T, an optimal solution to the (u, x)-, (u, x, s)- or $(u, 0)$-problem can be obtained by a combination of the optimal solutions to some subproblems. Suppose that node u has k children $c_1, c_2, \cdots, c_k$ as shown in Figure 13. Since $E_u = \{(u, p(u))\} \cup \bigcup_{i=1}^{k} E_{c_i}$, it follows that

$$\mathcal{C}(E_u) = \mathcal{C}(\{(u, p(u))\}) + \sum_{i=1}^{k} \mathcal{C}(E_{c_i}). \tag{10}$$

Note that $\mathcal{C}(\{(u, p(u))\})$ is essentially $d(u, p(u))$ times the number of link $(u, p(u))$'s occurrences in the interaction paths between all pairs of clients. Given a (u, x)-, (u, x, s)- or $(u, 0)$-problem, we show that there are a fixed number of $(u, p(u))$'s occurrences, which is independent of the T_u-client assignment.

In the (u, x)-problem, all clients in the entire tree T can be classified into three groups: $|T| - |T_u|$ clients are outside T_u and are assigned to servers outside T_u (abbreviated as Out-to-Out clients); x clients are inside T_u and are assigned to servers outside T_u (abbreviated as In-to-Out clients); and $|T_u| - x$ clients are inside T_u and are assigned to servers inside T_u (abbreviated as In-to-In clients). The interaction paths between Out-to-Out clients do not contain the link $(u, p(u))$. Similarly, the interaction paths between In-to-In clients do not contain the link $(u, p(u))$. Each interaction path between two In-to-Out clients or between an In-to-Out client and an In-to-In client contains the link $(u, p(u))$ twice. In addition, each interaction path between an Out-to-Out client and an In-to-In or In-to-Out client contains the link $(u, p(u))$ once. As a result, the total number of $(u, p(u))$'s occurrences is

$$\begin{aligned} &2\Big(\frac{x(x+1)}{2} + x \cdot (|T_u| - x)\Big) + |T_u| \cdot (|T| - |T_u|) \\ &= |T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T_u| - x^2 + x, \end{aligned} \tag{11}$$

which is a fixed number for a given (u, x)-problem.

Similarly, it can be derived that the numbers of $(u, p(u))$'s occurrences in the (u, x, s)- and $(u, 0)$-problems are respectively given by

$$|T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T| - 2x \cdot |T_u| - x^2 + x \tag{12}$$

and

$$|T_u| \cdot |T| - |T_u|^2, \tag{13}$$

which are also fixed numbers for the respective problems (please refer to Appendix D in the supplementary file for detailed derivations). Therefore, in the (u, x)-, (u, x, s)- and $(u, 0)$-problems, the objective of minimizing $\mathcal{C}(E_u)$ is equivalent to minimizing $\sum_{i=1}^{k} \mathcal{C}(E_{c_i})$.

Let $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ be the optimal T_u-client assignments to the (u, x)-, (u, x, s)- and $(u, 0)$-problems respectively. It follows from the above analysis that all T_{c_i}-client assignments used in $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ are also optimal assignments to their corresponding subproblems for respective subtrees T_{c_i}. Otherwise, if there exist better T_{c_i}-client assignments, then replacing the T_{c_i}-client assignments in $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ with these better T_{c_i}-client assignments reduces $\mathcal{C}(E_{c_i})$ and thus $\mathcal{C}(E_u)$, which contradicts to the optimality of $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$.

Now we show how to compute the optimal assignments. We start by solving the client assignment problem for the case where servers are located at the leaf nodes only.

To facilitate presentation, for each node u in T, we represent the optimal assignments $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ as the sets of pairs of clients and their assigned servers (i.e., $\{(c, s) \mid c \in T_u,\ s \in S,\ c \text{ is assigned to } s\}$). We also let $l(u, x)$, $l(u, x, s)$ and $l(u, 0)$ represent the values of $\mathcal{C}(E_u)$ for the optimal assignments $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ respectively.

The optimization problems are trivial if u is a leaf node in T. If $u \notin S$ (i.e., there is no server at u), the client at u must be assigned to a server outside T_u. Thus, among all optimization problems of T_u, only the $(u, 1)$-problem is a valid problem. $A(u, 1)$ assigns u to a server outside T_u, and $l(u, 1) = (|T| + 1) \cdot d(u, p(u))$ is a fixed value regardless of u's assigned server. If $u \in S$ (i.e., there is a server at u), then only the (u, x, u)- and $(u, 0)$-problems are valid. It is easy to obtain $A(u, x, u) = \{(u, u)\}$, $l(u, x, u) = \big(x(x+1) + 2x(|T| - x - 1) + (|T| - 1)\big) \cdot d(u, p(u))$; and $A(u, 0) = \{(u, u)\}$, $l(u, 0) = (|T| - 1) \cdot d(u, p(u))$.

For each non-leaf node u in T, we must examine the solutions to all possible subproblems in order to determine which solutions to use in the optimal assignments $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$.

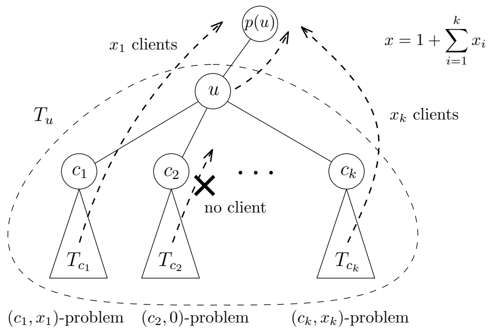


Fig. 13. A recursive solution to the (u, x)-problem.

In the (u, x)-problem, x clients inside T_u are assigned to servers outside T_u. Since the paths from these clients to their assigned servers all go through node u, according to Theorem 2, these x clients must be assigned to the same server outside T_u, and the client at node u must be one of these x clients. In addition, no client can be assigned to any other server through paths involving u. Thus, in the optimal assignment $A(u, x)$, no client outside each subtree T_{c_i} is assigned to servers inside T_{c_i}. As shown in Figure 13, for each subtree T_{c_i} $(1 \le i \le k)$, if the T_{c_i}-client assignment used in $A(u, x)$ does not assign any client inside T_{c_i} to servers outside T_{c_i}, then it must be an optimal solution to the $(c_i, 0)$-problem. Otherwise, if the T_{c_i}-client assignment used in $A(u, x)$ assigns x_i $(x_i \ge 1)$ clients inside T_{c_i} to servers outside T_{c_i}, it must be an optimal solution to the (c_i, x_i)-problem. Moreover, based on Theorem 2, these x_i clients must be part of the x clients inside T_u that are assigned to a server outside T_u. Therefore, to solve the (u, x)-problem, we need to consider the $(c_i, 0)$-problem and the (c_i, x_i)-problems $(1 \le x_i \le |T_{c_i}|)$ for each subtree T_{c_i}. All $A(c_i, x_i)$s selected to be used in $A(u, x)$ must satisfy $\sum_{i=1}^{k} x_i = x - 1$, where "1" represents the client at u.

Therefore, following equation (10) and expression (11), the recurrence for the (u, x)-problem is given by

$$l(u,x) = (|T_u|\cdot|T| - |T_u|^2 + 2x\cdot|T_u| - x^2 + x)\cdot d(u,p(u)) + \min_{\substack{0\le x_i\le|T_{c_i}| \\ \sum_{i=1}^{k} x_i = x-1}} \{\sum_{i=1}^{k} l(c_i,x_i)\}. \quad (14)$$

It is easy to see that for any node u and number x, the value of $l(u,x)$ is independent of which server outside T_u the x clients are assigned to. Thus, we represent the assigned server of the x clients by a special symbol $\mathbb{O}$ in $A(u,x)$. Let x_i^* denote the value of x_i that produces the minimum $\sum_{i=1}^{k} l(c_i,x_i)$ in the above recurrence. Then, the optimal assignment $A(u,x)$ is given by

$$A(u,x) = \{(u,\mathbb{O})\} \cup \bigcup_{i=1}^{k} A(c_i,x_i^*).$$

The symbol $\mathbb{O}$ would be replaced by a specific server later in the recurrences for the (u,x,s)- and $(u,0)$-problems (please refer to Appendix E for details).

If all possible values of x_is are examined exhaustively in the computation of recurrence (14), the computation still has an exponential asymptotic complexity. The term $\min_{0\le x_i\le|T_{c_i}|,\sum_{i=1}^{k} x_i = x-1}\{\sum_{i=1}^{k} l(c_i,x_i)\}$ in recurrence (14) can be computed efficiently as follows. Let $f_1(j,x')$ be the minimum value of $\sum_{i=1}^{j} l(c_i,x_i)$ subject to the constraint $\sum_{i=1}^{j} x_i = x'$, i.e.,

$$f_1(j,x') = \min_{\substack{0\le x_i\le|T_{c_i}| \\ \sum_{i=1}^{j} x_i = x'}} \{\sum_{i=1}^{j} l(c_i,x_i)\},$$

where $1 \le j \le k$, and $0 \le x' \le x-1$. It is obvious that

$$\begin{cases} f_1(1,x') = l(c_1,x'), \\ f_1(j,x') = \min\limits_{\substack{0\le x_j\le \\ \min(x',|T_{c_j}|)}} \{f_1(j-1,x'-x_j) + l(c_j,x_j)\}. \end{cases}$$

Thus, starting from $j = 1$, we can compute all $f_1(j,x')$s in an increasing order of j. Then, $l(u,x)$ can be computed with $f_1(k,x-1)$.

The optimal solutions to the (u,x,s)- and $(u,0)$-problems are computed from the optimal solutions to the (c_i,x_i)-, (c_i,x_i,s)- and $(c_i,0)$-problems in similar ways. Please refer to Appendix E in the supplementary file for detailed derivations of the recurrences.

Starting from the subproblems where u is a leaf node, we can solve all (u,x)-, (u,x,s)- and $(u,0)$-optimization problems by a postorder traversal of T. As analyzed in Appendix E in the supplementary file, the total time complexity of our algorithm for computing an optimal client assignment in the entire tree T is $O(K\cdot|S|\cdot|T|^3)$, where K is the maximum degree of all nodes in T, and $|S|$ is the number of servers.

Note that the optimal client assignments to all problems of each subtree T_u are computed from the optimal assignments in subtrees rooted at u's children. Therefore, the proposed dynamic programming algorithm can be implemented in a distributed manner by having each node compute the optimal client assignments in its local subtree and send them to its parent on a level-by-level basis.

The above dynamic programming solution can be directly extended to the general case where servers can be located at any nodes in the tree. Please refer to Appendix F in the supplementary file for detailed discussions.

VII. Related Work

The salient feature of real-time mutual interactions makes the client assignment in DIAs fundamentally different from that in web content distribution. In web content distribution, since web clients simply download contents from web servers, it is straightforward to direct web clients to their nearest servers for minimizing the access latency [10], [16]. In contrast, each client in DIAs connects to one server through which it interacts with all the other clients, and the interaction time between clients includes both the client-to-server latency and inter-server latency. Previous studies on client assignment in distributed virtual environments considered only the client-to-server latency as the objective of optimization [21], [22], [23]. As have been shown by our results, reducing the client-to-server latency alone is not effective for improving interactivity.

Where to place servers in the network is another important issue relevant to interactivity enhancement [10]. When the client-to-server latency is the only optimization objective, server placement is related to several graph theoretic problems: the facility location problem, and the k-median and k-center problems. These problems are all NP-hard and have been well studied in the literature. For the facility location problem, Hochbaum [24] first gave a greedy algorithm with an approximation ratio of $O(\log n)$. Korupolu *et al.* [25] proposed a local search heuristic that has a constant approximation ratio of $5+\varepsilon$, which has been subsequently improved to 3 [26]. Aggarwal *et al.* [27] presented a 3-approximation algorithm assuming that each facility has the same limited service capacity. For the k-median problem, it has been shown that polynomial optimal solutions exist on tree graphs [28]. For general graphs, a naive greedy algorithm has an approximation ratio of $\Omega(n)$ [29]. Local search heuristics have been proposed with the approximation ratio of $3+\varepsilon$ [26]. For the k-center problem, there is a 2-approximation algorithm. This ratio cannot be further improved unless $P = NP$ [30]. Different from above work, Lee *et al.* [9] studied how to place servers for online games taking into account the inter-server latency. However, no approximation ratio is given for their proposed algorithm. Complementary to server placement, we aim to assign clients to appropriate servers for maximizing interactivity given a set of servers placed in the network. Our work in this paper shows that finding an optimal client assignment is a challenging task, even under carefully planned server placements.

The formulation of our client assignment problem is similar to that of the fixed-hub single allocation problem that arose in transportation systems. In these systems, hubs are placed in the network and used as transshipment points. The traffic in the network may flow between the non-hub nodes through hubs and/or between hubs. The fixed-hub single allocation problem asks: given a set of fixed hubs, how to assign each node to one hub to minimize the total cost of traffic routing in the network. Campbell [17] proposed two heuristic solutions called *MAXFLO* (which we have included in our evaluation) and *ALLFLO* (which is an exponential time algorithm). Iwasa *et al.* [31] presented a 3-approximation algorithm that is similar to our *Nearest-Assignment* algorithm. They further provided a 2-approximation algorithm by using a linear programming relaxation and a randomized rounding technique. However, the size of the linear programming relaxation is large and the algorithm is only suitable for small networks. In addition, it is a centralized algorithm that is difficult to adapt to dynamic networks.

VIII. Conclusion

In this paper, we have investigated the client assignment problem for enhancing the interactivity performance of DIAs. We have formulated the problem as a combinational optimization problem and proved that it is NP-complete. Three heuristic algorithms have been proposed with their approximation ratios theoretically analyzed and their performance experimentally evaluated using real Internet latency data. The results show that: (1) the proposed *Modify-Assignment* and *Distributed-Modify-Assignment* algorithms significantly outperform the intuitive *Nearest-Assignment* algorithm for different server placements; and (2) *Distributed-Modify-Assignment* adapts well to the dynamics of both client participation and network latency. We have also developed a polynomial-time algorithm for computing the optimal client assignment in tree topologies.

References

[1] L. Zhang and X. Tang, "Client assignment for improving interactivity in distributed interactive applications," in *Proc. IEEE INFOCOM 2011*.

[2] S. Webb, S. Soh, and W. Lau, "Enhanced mirrored servers for network games," in *Proc. ACM SIGCOMM NetGames 2007*, pp. 117–122.

[3] L. Ahmad, A. Boukerche, A. Al Hamidi, A. Shadid, and R. Pazzi, "Web-based e-learning in 3D large scale distributed interactive simulations using HLA/RTI," in *Proc. IEEE IPDPS 2008*, pp. 1–4, 2008.

[4] A. Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, "CoMaya: incorporating advanced collaboration capabilities into 3D digital media design tools," in *Proc. ACM CSCW 2008*, pp. 5–8.

[5] F. Safaei, P. Boustead, C. Nguyen, J. Brun, and M. Dowlatshahi, "Latency-driven distribution: infrastructure needs of participatory entertainment applications," *IEEE Commun. Mag.*, vol. 43, no. 5, pp. 106–112, 2005.

[6] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system," tech. rep., University of Michigan, 2001.

[7] L. Briceño, H. Siegel, A. Maciejewski, Y. Hong, B. Lock, M. Teli, F. Wedyan, C. Panaccione, C. Klumph, K. Willman, and C. Zhang, "Robust resource allocation in a massive multiplayer online gaming environment," in *Proc. 4th International Conference on Foundations of Digital Games*, pp. 232–239, 2009.

[8] C. Jay, M. Glencross, and R. Hubbold, "Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment," *ACM Trans. Comput.-Hum. Interact.*, vol. 14, no. 2, 2007.

[9] K. Lee, B. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," *Computer Networks*, vol. 49, no. 1, pp. 84–102, 2005.

[10] E. Cronin, S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 7, pp. 1369–1382, 2002.

[11] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman and Company, Calif, 1979.

[12] K. Gummadi, S. Saroiu, and S. Gribble, "King: Estimating latency between arbitrary Internet end hosts," in *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurment*, pp. 5–18, 2002.

[13] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality and routing policy violations in the internet," in *Proc. PAM 2009*, pp. 45–54, 2009.

[14] P. Morillo, J. Orduna, M. Fernandez, and J. Duato, "Improving the performance of distributed virtual environment systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 7, pp. 637–649, 2005.

[15] "The meridian latency data set." Available at: http://www.cs.cornell.edu/People/egs/meridian/.

[16] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *Proc. IEEE INFOCOM 2001*, pp. 1587–1596.

[17] J. Campbell, "Hub location and the p-hub median problem," *Operations Research*, vol. 44, no. 6, pp. 923–935, 1996.

[18] "Planetlab all-pairs-pings." Available at: http://pdos.lcs.mit.edu/~strib/.

[19] M. Hosseini, D. Ahmed, S. Shirmohammadi, and N. Georganas, "A survey of application-layer multicast protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 58–74, 2007.

[20] L. Lao, J. Cui, M. Gerla, and S. Chen, "A scalable overlay multicast architecture for large-scale applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 449–459, 2007.

[21] D. Ta and S. Zhou, "A network-centric approach to enhancing the interactivity of large-scale distributed virtual environments," *Computer Communications*, vol. 29, no. 17, pp. 3553–3566, 2006.

[22] D. Ta and S. Zhou, "A two-phase approach to interactivity enhancement for large-scale distributed virtual environments," *Computer Networks*, vol. 51, no. 14, pp. 4131–4152, 2007.

[23] S. Webb and S. Soh, "Adaptive client to mirrored-server assignment for massively multiplayer online games," in *Proc. MMCN*, 2008.

[24] D. Hochbaum, "Heuristics for the fixed cost median problem," *Mathematical programming*, vol. 22, no. 1, pp. 148–162, 1982.

[25] M. Korupolu, C. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," *Journal of algorithms*, vol. 37, no. 1, pp. 146–188, 2000.

[26] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on Computing*, vol. 33, no. 3, pp. 544–562, 2004.

[27] A. Aggarwal, L. Anand, M. Bansal, N. Garg, N. Gupta, S. Gupta, and S. Jain, "A 3-approximation for facility location with uniform capacities," in *Proc. IPCO 2010*, pp. 149–162, 2010.

[28] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs," *Operations Research Letters*, vol. 19, no. 2, pp. 59–64, 1996.

[29] M. Chrobak, C. Kenyon, and N. Young, "The reverse greedy algorithm for the metric k-median problem," *Information Processing Letters*, vol. 97, no. 2, pp. 68–72, 2006.

[30] T. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[31] M. Iwasa, H. Saito, and T. Matsui, "Approximation algorithms for the single allocation problem in hub-and-spoke networks and related metric labeling problems," *Discrete Applied Mathematics*, vol. 157, no. 9, pp. 2078–2088, 2009.

Lu Zhang (S'11) received the BEng degree in computer science and engineering from University of Science and Technology of China. He is currently a PhD student in computer science at Nanyang Technological University, Singapore. His research interests include distributed systems, approximation algorithms and resource provisioning in distributed networks.

Xueyan Tang (M'04–SM'09) received the BEng degree in computer science and engineering from Shanghai Jiao Tong University in 1998, and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2003. He is currently an associate professor in the School of Computer Engineering at Nanyang Technological University, Singapore. His research interests include distributed systems, mobile and pervasive computing, wireless sensor networks, Web and Internet.