The Client Assignment Problem for Continuous Distributed Interactive Applications

Lu Zhang and Xueyan Tang
School of Computer Engineering
Nanyang Technological University
Singapore 639798
Email: {zh0007lu, asxytang}@ntu.edu.sg

***Abstract*—Interactivity is a primary performance measure for distributed interactive applications (DIAs) that enable participants at different locations to interact with each other in real time. Wide geographical spreads of participants in large-scale DIAs necessitate distributed deployment of servers to improve interactivity. In a distributed server architecture, the interactivity performance depends on not only client-to-server network latencies but also inter-server network latencies as well as synchronization delays to meet the consistency and fairness requirements of DIAs. All of these factors are directly affected by how the clients are assigned to the servers. In this paper, we investigate the problem of effectively assigning clients to servers for maximizing the interactivity of DIAs. We focus on continuous DIAs that change their states not only in response to user operations but also due to the passing of time. We analyze the minimum achievable interaction time for DIAs to preserve consistency and provide fairness among clients, and formulate the client assignment problem as a combinational optimization problem. We prove that this problem is NP-complete. Four heuristic assignment algorithms are proposed and evaluated using real Internet latency data. The experimental results show that our proposed greedy algorithm generally produces near optimal interactivity and significantly reduces the interaction time between clients compared to the intuitive algorithm that assigns each client to its nearest server.**

I. Introduction

Distributed Interactive Applications (DIAs) are networked systems that enable participants at different locations to interact with one another in real time. Since DIAs are human-in-the-loop applications, improving the interactivity of DIAs is critically important for supporting graceful interactions among participants. The interactivity performance is normally characterized by the duration from the time when a participant issues an operation to the time when the effect of the operation is presented to the same participant or other participants [15]. This duration is referred to as the *interaction time* between participants. Network latency is known as a major barrier to achieving high interactivity [9]. Increasing geographical spreads of participants in large-scale DIAs is making distributed server deployment vital for combating the network latency. Latency-driven distribution of servers is essential even when there are no limitations on the availability of server resources at one location [21].

In a distributed server architecture, the state of a DIA (such as the virtual world in a multi-player online game) is often replicated across a group of geographically distributed servers [3], [7]. As shown in Fig. 1, each participant, known as a client, is assigned to one server and connects to the server for sending user-initiated operations and receiving updates of the application state. When issuing an operation, a client first sends the operation to its assigned server. Then, the server forwards the operation to all the other servers. On receiving the operation, each server calculates the new state of the application and sends a state update to all the clients assigned to it. Thus, the clients interact with one another through their assigned servers. The interaction time between any pair of clients must include 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. In addition, the interaction time is also influenced by the consistency and fairness requirements of DIAs. Consistency means that shared common views of the application state must be created among all clients to support meaningful interactions [9]. Fairness, on the other hand, is to ensure that all clients have the same chance of participation regardless of their network conditions [5], [17]. Maintaining consistency and fairness in DIAs usually introduces artificial synchronization delays in the interactions among clients [5], [8], [12], [16], [18]. These synchronization delays are also dependent on the assignment of clients to servers. Therefore, how to assign the clients to the servers in DIAs is of crucial importance to their interactivity performance.

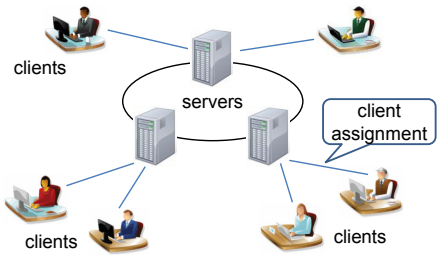


Figure 1. Distributed server architecture.

In this paper, we investigate the problem of effectively assigning clients to servers for maximizing the interactivity

of DIAs. We focus on continuous DIAs that change their states not only in response to user-initiated operations but also due to the passing of time [18]. Examples of continuous DIAs include distributed virtual environments, distributed interactive simulations and multi-player online games.

We start by mathematically modeling the interactivity performance of continuous DIAs under the consistency and fairness requirements. For any given client assignment, we analyze the minimum achievable interaction time for DIAs to preserve consistency and provide fairness among clients. Based on the analysis, we formulate the client assignment problem as a combinational optimization problem and prove that it is NP-complete. Several heuristic assignment algorithms are then proposed and experimentally evaluated using real Internet latency data. The results show that the proposed greedy assignment algorithms significantly reduce the interaction time between clients compared to the intuitive *Nearest-Server Assignment* algorithm that assigns each client to its nearest server. The interactivity produced by the greedy assignment algorithms is generally close to the optimum.

The rest of this paper is organized as follows. Section II analyzes the minimum achievable interaction time and formulates the client assignment problem. Section III presents the NP-completeness result. Section IV proposes four heuristic client assignment algorithms for improving the interactivity of DIAs. The experimental setup and results are discussed in Section V. Section VI summarizes the related work. Finally, Section VII concludes the paper.

II. Problem Formulation

A. System Model

We model the underlying network supporting a DIA by a graph $G = (V, E)$, in which 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. Let $S \subseteq V$ be the set of servers in the network and $C \subseteq V$ be the set of clients. Each client needs to be assigned to a server in order to send user operations and receive state updates. A client assignment is a mapping from C to S, where for each client $c \in C$, we denote by $s_A(c) \in S$ as the server that client c is assigned to.

The clients interact with one another through their assigned servers. Specifically, when a client c_i issues an operation, the effect of the operation is presented to another client c_j through the following process. First, c_i 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, possibly after some artificial synchronization delay, and then delivers the resultant state update to c_j. In the above interaction process, 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 are involved. Similarly, if c_j issues an operation, the same three paths in the network 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. For example, in the network of Fig. 2, suppose clients c_1 and c_2 are assigned to server s_1, and client c_3 is assigned to server s_2. Then, the interaction path between c_1 and c_2 is indicated by the dotted line, and the interaction path between c_1 and c_3 is indicated by the dashed line.

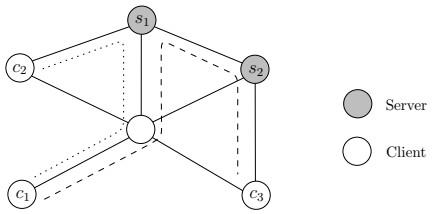


Figure 2. Interaction paths in a network.

The length of the interaction path between two clients c_i and c_j represents the network latency involved in their interaction. To facilitate presentation, we extend the distance 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. Then, the length of the interaction path between c_i and c_j is given by $d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)$. Note that the length of the interaction path from a client c_i to itself is $2d(c_i, s_A(c_i))$, i.e., the round-trip time between c_i and its assigned server $s_A(c_i)$, and represents the network latency involved for c_i to see the effect of its own operation.

B. Consistency and Fairness Models

In continuous DIAs, the progress of the application state is typically measured by the time elapsed since the initial state of the application [18]. We shall call it the *simulation time*. For instance, the simulation time of a multi-player online game records the time elapsed in its virtual world. In the distributed server architecture, each server and client has a copy of the application state and its associated simulation time. The simulation times of all servers and clients should advance at the same rate. However, they do not have to be synchronized, i.e., their readings do not have to be the same at the same wall-clock time. Normally, the simulation time of a client lags behind the simulation time of its assigned server due to the network latency of delivering state updates from the server to the client.

The consistency requirement for continuous DIAs is to ensure that all clients share the same view of the application state when their respective simulation times reach the same value. This is automatically guaranteed among the clients assigned to the same server because they all inherit the application state from their assigned server through state updates. Nevertheless, the application state seen by the clients assigned to different servers may not be identical at

the same simulation time if the application states maintained by their assigned servers are not consistent. Since the state of a continuous DIA changes due to both user operations and time passing, to ensure consistency among the application states at the servers, each user operation must be executed by all servers at the same simulation time.

The fairness requirement is to ensure that all clients have the same chance of participation regardless of their network conditions. This is particularly important for applications where users compete with each other. In essence, fairness is concerned with the order of executing user operations [17]. For example, a participant would gain an unfair advantage in an air combat game if an action taken by him at a later time takes effect before another action taken by a different participant at an earlier time. To guarantee fairness in continuous DIAs, the order of operation execution must be the same as the operation issuance order at the clients based on the simulation time. In addition, the time interval between the issuances of two operations in terms of simulation time must also be preserved between the executions of these operations. This entails executing each operation at the server at a simulation time that is of a constant lag behind the simulation time of the operation issuance.

Integrating both consistency and fairness requirements, we get the following criterion: all operations must be executed by all servers at simulation times that is of a constant lag behind the operation issuances. In the following, we analyze the minimum achievable interaction time for continuous DIAs to meet this criterion.

C. Minimum Achievable Interaction Time

Let δ denote the constant lag for operation execution. We first show that the average interaction time between all client pairs is equal to δ. For each pair of nodes $u, v \in C \cup S$ in the network, we denote by $\Delta_{u,v}$ the offset of node u's simulation time relative to node v's simulation time (a positive offset means that u's simulation time is ahead of v's simulation time). Since the simulation times of all servers and clients advance at the same rate, the offset $\Delta_{u,v}$ for each node pair u, v is a constant. Obviously, we have $\Delta_{u,v} + \Delta_{v,u} = 0$, and $\Delta_{u,v} + \Delta_{v,w} = \Delta_{u,w}$.

Consider the operation that is issued at simulation time t. Since all servers execute the operation at simulation time $t+\delta$, due to state inheritance, all clients should see the operation taking effect when their simulation times reach $t+\delta$. When a client c_j's simulation time reaches $t+\delta$, the simulation time at c_i is $t+\delta+\Delta_{c_i,c_j}$. Thus, the interaction time for c_j to see the effect of c_i's operation is $t+\delta+\Delta_{c_i,c_j}-t = \delta+\Delta_{c_i,c_j}$. Similarly, if client c_j issues an operation when its simulation time is t, the interaction time for c_i to see the effect of c_j's operation is $t+\delta+\Delta_{c_j,c_i}-t = \delta+\Delta_{c_j,c_i}$. As a result, the average interaction time between any two clients c_i and c_j is $\frac{1}{2}(\delta+\Delta_{c_i,c_j}+\delta+\Delta_{c_j,c_i}) = \delta$. So, the average interaction time between all client pairs is also equal to δ. Therefore, minimizing the average interaction time is equivalent to minimizing the constant lag δ.

To make operation execution and state updates feasible, δ must satisfy the following constraints:

(i) When any client issues an operation at its simulation time t, all servers are able to receive the operation before their respective simulation times reach $t+\delta$.

(ii) All clients are able to receive the resultant state update of executing the operation before their respective simulation times reach $t+\delta$.

Next, we show that the minimum possible value of δ under the above constraints is the maximum length of interaction paths between all client pairs.

We start by formulating constraints (i) and (ii) mathematically. Suppose an operation o is issued by a client c at its simulation time t. Since o is first sent to c's assigned server $s_A(c)$ and then delivered from $s_A(c)$ to all the other servers, the total network latency for any server s to receive operation o is $d(c, s_A(c)) + d(s_A(c), s)$.[1] Based on constraint (i), each server s should receive o before its simulation time reaches $t+\delta$. Thus, we have

$$\forall s, \quad t + d(c, s_A(c)) + d(s_A(c), s) + \Delta_{s,c} \leq t + \delta.$$

Constraint (i) stipulates that the above inequalities should hold for any client c. Therefore, constraint (i) can be formulated as

$$\forall c, s, \quad d(c, s_A(c)) + d(s_A(c), s) + \Delta_{s,c} \leq \delta. \qquad (1)$$

After operation o is executed by all servers when their respective simulation times reach $t+\delta$, the resultant state update should be delivered to all clients. Suppose each server sends the state update to its clients immediately after executing o. Then, for each client c, its simulation time when it receives the state update is $t+\delta+d(s_A(c), c)+\Delta_{c,s_A(c)}$, which, according to constraint (ii), should be before the simulation time of client c reaches $t+\delta$. Therefore, constraint (ii) can be formulated as

$$\forall c, \quad d(s_A(c), c) + \Delta_{c,s_A(c)} \leq 0. \qquad (2)$$

We now derive the minimum possible value of δ from inequalities (1) and (2). For any client pair c_i and c_j, based on (1), we have

$$d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + \Delta_{s_A(c_j),c_i} \leq \delta.$$

Based on (2), we have

$$d(s_A(c_i), c_i) + \Delta_{c_i,s_A(c_i)} \leq 0$$

By adding up the above two inequalities, it follows that

$$2d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + \Delta_{s_A(c_j),s_A(c_i)} \leq \delta. \quad (3)$$

[1] Since $d(s_A(c), s_A(c)) = 0$, we can use this expression to represent the network latency for server $s_A(c)$ to receive operation o as well.

Similarly, we have

$$2d(c_j, s_A(c_j)) + d(s_A(c_j), s_A(c_i)) + \Delta_{s_A(c_i),s_A(c_j)} \leq \delta. \quad (4)$$

By adding up (3) and (4), we have

$$d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(c_j, s_A(c_j)) \leq \delta. \quad (5)$$

The expression on the left side of inequality (5) is the length of the interaction path between clients c_i and c_j. Since c_i, c_j can be any pair of clients, (5) indicates that δ should be no less than the maximal length of interaction paths between all client pairs. Denote by D the maximum length of interaction paths between all client pairs, i.e.,

$$D = \max_{c_i,c_j \in C} \{d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)\}.$$

Then, the minimum possible value of δ is D.

Finally, we show that the minimum possible value of δ derived above is achievable by properly setting the offsets between simulation times at the servers and clients. We first synchronize the simulation times at all clients, i.e., setting $\Delta_{c,c'} = 0$ for all client pairs c, c'. Then, the values of $\Delta_{s,c}$ between a given server s and all clients c are set as follows:

$$\Delta_{s,c} = D - \max_{c' \in C} \{d(c', s_A(c')) + d(s_A(c'), s)\},$$

where the second term is the longest distance from server s to all clients through their assigned servers. Note that, in this setting, the simulation times of different servers may not be synchronized. We now show that the above $\Delta_{c,c'}$ and $\Delta_{s,c}$ values together with setting $\delta = D$ satisfy constraints (i) and (ii). For constraint (i), for each client c and server s, we have

$$\begin{aligned} & d(c, s_A(c)) + d(s_A(c), s) + \Delta_{s,c} \\ & = D + d(c, s_A(c)) + d(s_A(c), s) \\ & \quad - \max_{c' \in C} \{d(c', s_A(c')) + d(s_A(c'), s)\} \\ & \leq D = \delta. \end{aligned}$$

For constraint (ii), for each client c, we have

$$\begin{aligned} & d(s_A(c), c) + \Delta_{c,s_A(c)} \\ & = d(s_A(c), c) - \Delta_{s_A(c),c} \\ & = \max_{c' \in C} \{d(c', s_A(c')) + d(s_A(c'), s_A(c))\} + d(s_A(c), c) - D \\ & = \max_{c' \in C} \{d(c', s_A(c')) + d(s_A(c'), s_A(c)) + d(s_A(c), c)\} - D \\ & \leq 0. \end{aligned}$$

As a result, both constraints are satisfied. Therefore, with above setting of $\Delta_{c,c'}$ and $\Delta_{s,c}$ values, it is feasible to make the average interaction time δ equal to the maximum length D of interaction paths between all client pairs. Since the simulation times of all clients in this setting are synchronized, the interaction times between all client pairs are in fact the same, which are all equal to D.

D. Problem Statement

The above analysis shows that given a client assignment, the minimum achievable interaction time meeting the consistency and fairness requirements is the maximum length D of interaction paths between all client pairs. Therefore, the client assignment problem for maximizing the interactivity of continuous DIAs 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 that minimizes the maximum length of interaction paths between all client pairs, i.e.,

minimize

$$\max_{c_i,c_j \in C} \{d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)\}.$$

E. Further Considerations

In this paper, we focus on reducing the network latencies and the associated synchronization delays involved in the interaction between clients. Thus, the above problem formulation has not taken into consideration the processing delays at the servers. In general, the processing delays at the servers are easier to improve than the network latencies [14]. A busy server can always be better provisioned (e.g., by forming a server cluster) to meet the capacity requirements and reduce the processing delay. We shall discuss in Section IV-E how to deal with server capacity constraints in our proposed client assignment algorithms if server capacities are limited.

There may exist jitter in the network. Jitter refers to the variability of network latency. In the presence of jitter, longer synchronization delay would be required to cater for the variation in network latency in order to guarantee consistency and fairness. Our formulation of the client assignment problem is also valid in dealing with network jitter in that the length $d(u, v)$ of each link (u, v) can be set to any percentile of the network latency to cater for its variability to a required extent. At one extreme, setting $d(u, v)$ to the maximum possible network latency between nodes u and v would guarantee that each operation is received by all servers before its execution, thereby ensuring consistency and fairness. But this strategy may considerably degrade interactivity at large jitter. Thus, a real-world system often models a certain high percentile (e.g., 90th percentile) of the network latency to significantly reduce the chance for inconsistency and unfairness to arise [8], [17]. When inconsistency does occur due to jitter, the application state can be repaired using synchronization mechanisms such as timewarp [18] and Trailing State Synchronization (TSS) [8]. Repairing the application state, however, may create artifacts that disturb the user behavior. For instance, an artifact in an online game could mean that an opponent that has been beaten in a fight stands up again and continues to fight. Therefore, the extent to which the variability of network

latency is catered reflects a trade-off among interactivity, consistency and fairness. Selecting an appropriate percentile of the network latency to model based on the application needs is beyond the scope of this paper.

III. NP-Completeness Results

Finding a client assignment that minimizes the maximum length of interaction paths is a challenging task because the path length comprises of both client-to-server latencies and inter-server latencies. An intuitive client assignment is to assign each client to its nearest server [16], [26]. While this assignment reduces the client-to-server latencies, it could significantly increase the latencies between the assigned servers of different clients, and thus make the interactivity far worse than optimum as shall be shown by our experimental results in Section V. On the other hand, assigning all clients to a single server eliminates the contribution of inter-server latencies towards the lengths of interaction paths, but may remarkably increase the latencies between clients and their assigned server. In this section, we analyze the hardness of the client assignment problem. We prove that the problem is NP-complete.

Theorem 1: 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 an integer bound L. The length of the interaction path between each pair of clients can be computed in polynomial time. Thus, computing the maximum length of interaction paths between all client pairs and comparing it with the bound L can be done 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 minimum set cover problem which is known to be NP-complete [11]. The decision version of the minimum set cover problem is defined as follows: given a finite set P and a collection $\mathbb{Q}$ of its subsets, and a positive integer $K \leq |\mathbb{Q}|$, find out whether $\mathbb{Q}$ contains a set cover for P of size at most K, i.e., whether there exists a subset $\mathbb{Q}' \subseteq \mathbb{Q}$ with $|\mathbb{Q}'| \leq K$ such that $\bigcup_{Q \in \mathbb{Q}'} Q = P$.

Let R be an instance of the minimum set cover problem. Suppose that set P contains n elements $p_1, p_2, \cdots, p_n$ and collection $\mathbb{Q}$ contains m subsets $Q_1, Q_2, \cdots, Q_m$. We first construct a network $G = (V, E)$ where V consists of n clients and $m \cdot K$ servers (see Fig. 3). The n clients are $c_1, c_2, \cdots, c_n$, and each client c_i corresponds to one element p_i in set P. The servers are divided into K groups with each group containing m servers, i.e.,

$$s_1^1, s_2^1, \cdots, s_m^1; s_1^2, s_2^2, \cdots, s_m^2; \cdots; s_1^K, s_2^K, \cdots, s_m^K.$$

Each server corresponds to one subset in $\mathbb{Q}$, and the jth server s_j^l of each group l corresponds to subset Q_j. A client c_i and a server s_j^l are connected by a link if and only if element p_i belongs to subset Q_j. So, given j, a client c_i is either connected to the jth servers $s_j^1, s_j^2, \cdots, s_j^k$ of all groups or not connected to any of them. Besides, each server in one group is connected to all servers in other groups. Therefore, the set of links is

$$E = \{(c_i, s_j^l) | \forall l, p_i \in Q_j\} \cup \{(s_{j_1}^{l_1}, s_{j_2}^{l_2}) | \forall j_1, j_2, l_1 \neq l_2\}.$$

An instance T of the client assignment problem is then constructed on network G by setting the length of every link to 1, setting the bound L to 3, and assuming that messages are routed in the network by shortest path routing. Thus, the instance T is constructed in time polynomial to the size of instance R. In the following, we show that, $\mathbb{Q}$ contains a set cover $\mathbb{Q}'$ of size at most K for instance R if and only if there exists an assignment A with the maximum interaction path length at most 3 for instance T.

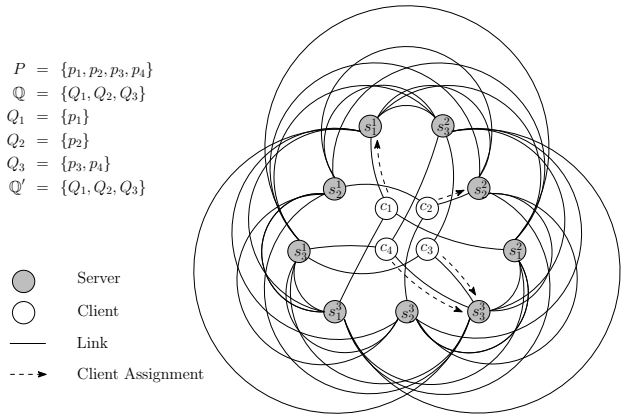


Figure 3. Example of instances R and T of the minimum set cover problem and the client assignment problem with $K = 3$.

Suppose there exists a set cover $\mathbb{Q}'$ with $|\mathbb{Q}'| \leq K$, then a client assignment is constructed in $|\mathbb{Q}'|$ steps as follows. At each step, we consider one subset $Q_j \in \mathbb{Q}'$ and try to assign the clients that correspond to Q_j's elements to the same server. We first choose an unused server group l such that no client has been assigned to any server in the group. Then, for each element $p_i \in Q_j$, client c_i is assigned to server s_j^l in group l if c_i has not been assigned to any server. Since we only assign clients to one server at each step, the total number of servers assigned clients is $|\mathbb{Q}'| \leq K$. Thus, it is guaranteed that an unused server group l can always be found at each step. In addition, since each element p_i belongs to at least one subset Q_j in $\mathbb{Q}'$, all clients must be assigned to servers after $|\mathbb{Q}'|$ steps. In the example of Fig. 3, the set cover $\mathbb{Q}' = \{Q_1, Q_2, Q_3\}$ contains three subsets of elements. Thus, the clients are assigned to three different servers s_1^1, s_2^2 and s_3^3. In the assignment that we constructed, each client c_i is assigned to one of the jth servers s_j^l of all groups only if p_i belongs to Q_j, which indicates that c_i is connected to s_j^l by a link. Thus, the distance between any

client and its assigned server is 1. Since in any group, at most one server is assigned clients, the distance between any two servers that are assigned clients is also 1. So, the length of interaction path between any pair of clients is 2 if they are assigned to the same server, and the length is 3 if they are assigned to different servers. Therefore, the maximum interaction path length in the constructed assignment is at most 3.

On the other hand, suppose there exists a client assignment with the maximum interaction path length not exceeding 3. Note that the distance between any client and server is at least 1. Thus, to make the length of interaction path between each client pair not exceeding 3, the distance between any two servers that are assigned clients cannot be longer than 1. This implies that in each group, there is at most one server that is assigned clients, since the distance between any two servers in the same group is 2. Thus, the total number of servers that are assigned clients is at most the number of server groups, i.e., K. We construct a set cover $\mathbb{Q}'$ based on this client assignment by selecting a subset Q_j if and only if there exists at least one jth server s_j^l that is assigned clients. Then, the size of $\mathbb{Q}'$ is at most K. In addition, note that the distance between a client and a server is 1 if there is a link connecting them and is at least 2 otherwise. If a client is assigned to a server to which it has no direct link, the length of the interaction path from the client to itself is at least 4. So, in the aforementioned assignment, each client must be assigned to a server to which it has a link. It follows that $\mathbb{Q}'$ must cover all elements in set P. Therefore, $\mathbb{Q}'$ is a set cover of size at most K.

Hence, the theorem is proven. ■

IV. Heuristic Algorithms

Since the client assignment problem is NP-complete, there may not exist a polynomial optimal solution for it. A brute-force algorithm is computationally expensive even with small numbers of clients and servers. In this section, we present four heuristic client assignment algorithms. The computation of these algorithms is simply based on the network latencies between clients and servers, which can be obtained with existing tools like ping and King [13]. Thus, these algorithms are generic and not tied to any particular routing strategy.

A. Nearest-Server Assignment

The first algorithm is called *Nearest-Server Assignment*, which intuitively assigns clients to their nearest servers. This algorithm can be implemented by having each client measure the network latencies between itself and all servers, and select the server with the lowest latency as its assigned server. The computational complexity for each client is hence $O(|S|)$. As discussed earlier, since *Nearest-Server Assignment* optimizes only client-to-server latencies and does not consider inter-server latencies, it cannot guarantee to produce good assignments. When assuming shortest path routing in the network, we can show that *Nearest-Server Assignment* has an approximation ratio 3.

Theorem 2: The maximum length of interaction paths between all client pairs in *Nearest-Server Assignment* is within three times of that in an optimal assignment.

Proof: Consider two clients u and v, whose nearest servers are n_u and n_v respectively. Suppose in an optimal assignment, client u is assigned to server o_u and client v is assigned to server o_v. Naturally, we have $d(u, n_u) \leq d(u, o_u)$ and $d(v, n_v) \leq d(v, o_v)$. Under shortest path routing, the distance function $d(u, v)$ satisfies the triangle inequality. Thus, it follows that:

$$d(n_u, o_u) \leq d(u, n_u) + d(u, o_u) \leq 2d(u, o_u),$$

and

$$d(n_v, o_v) \leq d(u, n_v) + d(u, o_v) \leq 2d(u, o_v).$$

In *Nearest-Server Assignment*, the length of the interaction path between u and v is $d(u, n_u) + d(n_u, n_v) + d(n_v, v)$. By the triangle inequality, $d(n_u, n_v)$ should not be greater than the length of the concatenation of the paths from n_u to o_u, from o_u to o_v, and from o_v to n_v. Therefore, we have

$$\begin{aligned}
&d(u, n_u) + d(n_u, n_v) + d(n_v, v) \\
&\leq d(u, n_u) + d(n_u, o_u) + d(o_u, o_v) + d(o_v, n_v) + d(n_v, v) \\
&\leq d(u, o_u) + 2d(u, o_u) + d(o_u, o_v) + 2d(o_v, v) + d(o_v, v) \\
&= 3d(u, o_u) + d(o_u, o_v) + 3d(o_v, v) \\
&\leq 3(d(u, o_u) + d(o_u, o_v) + d(o_v, v)).
\end{aligned}$$

This result shows that the interaction path for each client pair in *Nearest-Server Assignment* is within 3 times of that in an optimal assignment. The maximum interaction path length in *Nearest-Server Assignment* therefore has an approximation ratio of 3 to the optimum.

Hence, the theorem is proven. ■

The approximation ratio of 3 is tight. Fig. 4 gives an example in which there are two clients c_1, c_2 and three servers s, s_1, s_2. The distances between both clients and server s are a, and the distance from client c_1 to server s_1 and the distance from client c_2 to server s_2 are $a - \varepsilon$, where $\varepsilon > 0$. In *Nearest-Server Assignment*, client c_1 is assigned to server s_1, and client c_2 is assigned to server s_2. So, the maximum interaction path length is $6a - 4\varepsilon$. It is obvious that the optimal assignment should assign both clients to server s, and the maximum interaction path length in the optimal assignment is $2a$. The ratio between the two maximum interaction path lengths $6a - 4\varepsilon$ and $2a$ can be made arbitrarily close to 3 when ε approaches 0.

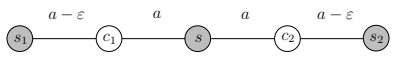


Figure 4. Example of the approximation ratio 3 of *Nearest-Server Assignment*.

B. Longest-First-Batch Assignment

The second algorithm is called *Longest-First-Batch Assignment*. The main idea is that, if a client c is assigned to a server s, assigning to s all clients that are not farther from s than c would not increase the maximum length of interaction paths. *Longest-First-Batch Assignment* first finds the nearest server for each client. Then, instead of assigning all clients to their nearest servers directly, the algorithm sorts the distances from the clients to their nearest servers, and assigns the clients iteratively. In each iteration, the algorithm finds an unassigned client c whose distance to its nearest server is the longest among all unassigned clients. Client c is then assigned to its nearest server, along with all unassigned clients that are not farther from this server than c. The algorithm terminates when all clients have been assigned to servers. Since the algorithm requires global knowledge about the distances between the clients and their nearest servers, it is better suited for centralized implementation.

In *Longest-First-Batch Assignment*, if a client is not assigned to its nearest server, it must not be the farthest client to its assigned server, and thus cannot be involved in the longest interaction path in the network. So, the longest interaction path in this assignment must connect two clients that are both assigned to their nearest servers. Thus, the maximum interaction path length in *Longest-First-Batch Assignment* cannot exceed that in *Nearest-Server Assignment* and is hence also within three times of the optimum. The approximation ratio of 3 is tight since the example shown in Fig. 4 applies here as well. Fig. 5 shows an example where *Longest-First-Batch Assignment* outperforms *Nearest-Server Assignment*. Here, c_1, c_2 are clients and s_1, s_2 are their respective nearest servers. In *Nearest-Server Assignment*, client c_1 is assigned to server s_1, and client c_2 is assigned to server s_2. The maximum interaction path length in this assignment is $5 + 4 + 3 = 12$. *Longest-First-Batch Assignment*, on the other hand, starts from client c_1, and assigns both clients to server s_1. Thus, the maximum interaction path length is $5 + 4 = 9$, which is shorter than that of *Nearest-Server Assignment*.

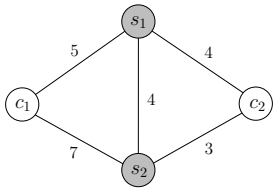


Figure 5. Example of the effectiveness of *Longest-First-Batch Assignment*.

In *Longest-First-Batch Assignment*, the first step that finds the nearest servers of all clients has a time complexity of $O(|C||S|)$, and sorting the distances between the clients and their nearest servers can be done in $O(|C| \log |C|)$ time. Each subsequent iteration has a worst-case time complexity of $O(|C|)$, which results in a total time complexity of $O(|C|^2)$ for all iterations. Thus, the overall time complexity of *Longest-First-Batch Assignment* is $O(|C|(|C| + |S|))$.

C. Greedy Assignment

The third algorithm *Greedy Assignment* adopts a greedy approach to assign clients to servers iteratively. It starts with an empty assignment and employs a similar idea to *Longest-First-Batch Assignment* when assigning clients. In each iteration, the algorithm considers all possibilities of assigning an unassigned client to a server. If a client c is selected to be assigned to a server s, and then all unassigned clients that are not farther from s than c are also assigned to server s. Let Δn be the number of new clients assigned to s, and Δl be the increase in the maximum interaction path length due to these new assignments. We define $\Delta l / \Delta n$ as a cost metric for selecting which client to be assigned to which server in an iteration, since we want to minimize the amortized increase in the maximum length of interaction paths for the new clients assigned. In each iteration, among all possible pairs of unassigned client and server (c, s), the pair resulting in the minimum cost $\Delta l / \Delta n$ is selected and the corresponding clients are assigned to s. The algorithm terminates when all clients have been assigned to servers.

To calculate Δn efficiently, the distances from all clients to each server s can be sorted in a list L_s in a preprocessing stage. This sorted list is incrementally updated by removing newly assigned clients at the end of every iteration. Then, Δn can be obtained directly from the index of the unassigned client in the list. On the other hand, Δl for assigning a new client c to a server s is calculated by comparing the current maximum interaction path length with the maximum length of the interaction paths from client c to all clients already assigned to servers. The latter is given by

$$\begin{aligned}&\max\big\{2d(c,s),\ \max_{b\in C'}\{d(c,s)+d(s,s_A(b))+d(s_A(b),b)\}\big\}\\ &=\max\big\{2d(c,s),\ d(c,s)+\max_{b\in C'}\{d(s,s_A(b))+d(s_A(b),b)\}\big\},\end{aligned}$$

where $2d(c, s)$ is the interaction path length from c to itself and C' is the set of clients already assigned to servers. For each server s, the term $\max_{b\in C'}\{d(s, s_A(b)) + d(s_A(b), b)\}$ is independent of client c, so its calculation can be shared among all unassigned clients. The pseudo code of *Greedy Assignment* is presented in Fig. 6. *Greedy Assignment* is also better suited for centralized implementation due to its need for global knowledge of the distances between clients and servers.

In the preprocessing stage, sorting lists L_s for all servers and calculating indexes of all clients in the lists can be done in $O(|S||C| \log |C|)$ time. Suppose all clients are assigned to servers in m iterations. To calculate the time complexity of each iteration, we divide it into three stages. Stage 1 (lines 9 to 21) is to find the pair of client and server with the minimum $\Delta l / \Delta n$. For each server in an iteration, the

1: $C' \leftarrow \emptyset$; //the set of clients already assigned to servers
2: $max\_len \leftarrow 0$; //the maximum interaction path length
3: **for all** $s \in S$ **do**
4: create a list L_s of all clients in C;
5: sort L_s according to $d(c, s)$ in ascending order;
6: **for** $i = 1$ to $|C|$ **do**
7: $index[s, L_s[i]] \leftarrow i$;
8: **while** $C' \neq C$ **do**
9: $min \leftarrow \infty$;
10: **for all** $s \in S$ **do**
11: $m \leftarrow \max_{b \in C'}\{d(s, s_A(b)) + d(s_A(b), b)\}$;
12: **for all** $c \in C - C'$ **do**
13: $\Delta n \leftarrow index[s, c]$;
14: $len \leftarrow \max\{2d(c, s),\ d(c, s) + m,\ max\_len\}$;
15: $\Delta l \leftarrow len - max\_len$;
16: $cost \leftarrow \frac{\Delta l}{\Delta n}$;
17: **if** $cost < min$ **then**
18: $min \leftarrow cost$;
19: $len^* \leftarrow len$;
20: $c^* \leftarrow c$;
21: $s^* \leftarrow s$;
22: $max\_len \leftarrow len^*$;
23: **for all** $c \in C - C'$ such that $d(c, s^*) \leq d(c^*, s^*)$ **do**
24: set $s_A(c) = s^*$;
25: $C' \leftarrow C' \cup \{c\}$;
26: **for all** $s \in S$ **do**
27: $nuc \leftarrow 0$; //the number of unassigned clients
28: **for** $i = 1$ to $|C|$ **do**
29: **if** $L_s[i] \in C - C'$ **then**
30: $nuc \leftarrow nuc + 1$;
31: $index[s, L_s[i]] \leftarrow nuc$;

Figure 6. The *Greedy Assignment* algorithm

time complexity of line 11 is $O(|C|)$. Then, the calculation of $\Delta l/\Delta n$ can be done in $O(1)$ time for each unassigned client and hence in $O(|C|)$ time for all unassigned clients. Thus, the total time complexity of stage 1 in each iteration is $O(|S||C|)$. Stage 2 (lines 22 to 25) is to add the new assignments of clients. The time complexity of this stage in each iteration is $O(|C|)$. Stage 3 (lines 26 to 31) is to update the indexes of unassigned clients in lists L_s by removing newly assigned clients. The time complexity of stage 3 is $O(|S||C|)$. So, the total time complexity for one iteration is $O(|S||C|)$. Therefore, the overall time complexity of *Greedy Assignment* is $O\big(|S||C| \log |C| + m|S||C|\big)$, or $O(|S||C|^2)$ in the worst case since $m \leq |C|$.

D. Distributed-Greedy Assignment

The fourth algorithm *Distributed-Greedy Assignment* is also a greedy heuristic, and it is performed in a distributed manner. *Distributed-Greedy Assignment* starts with an initial assignment, and continues to modify the assignment to reduce the maximum interaction path length, denoted by D, until it cannot be reduced further.

To calculate D of the initial assignment, each server measures its distances (network latencies) to all the other servers, and its distances to all clients that are assigned to it in the initial assignment. Then, each server s broadcasts to all the other servers the inter-server distances and its longest distance $l(s)$ to the clients assigned to it. Based on its received information, each server calculates D independently.

At each assignment modification, each server checks whether it is assigned any client that is involved in a longest interaction path. If so, the algorithm attempts to modify the assigned server of the client to reduce D. Suppose that server s is assigned a client c that is involved in a longest interaction path. First, server s broadcasts to all the other servers the identifier of c and the longest distance $l(s)$ to its assigned clients excluding c. On receiving the information, each of the other servers s' measures its distance (network latency) to c, and computes the maximum length of interaction paths involving c assuming c is assigned to it, which is given by $L(s') = \max_{s''}\{d(c, s') + d(s', s'') + l(s'')\}$. Then, each server s' sends the result $L(s')$ back to server s. If $\min_{s'} L(s') < D$, s reassigns c to the server s^* with the minimum $L(s^*)$. Finally, the new server of c updates its longest distance to its clients and broadcasts this distance to all the other servers if it is changed. If there is a unique longest interaction path and if $\min_{s'} L(s') < D$, D would decrease after reassigning c. However, if there are multiple longest interaction paths with equal lengths, reassigning c does not guarantee to reduce D. If D cannot be reduced after examining all clients involved in the longest interaction path(s), the algorithm terminates.

Note that a longest interaction path can involve two different clients. If two or more clients involved in longest interaction path(s) change their assigned server concurrently, the maximum interaction path length is not guaranteed to reduce because the calculation of each assignment modification is based on the assumption that the assigned servers of other clients remain unchanged. Thus, a concurrency control mechanism is required to prevent servers from performing assignment modifications simultaneously. In this way, each assignment modification can only reduce the maximum interaction path length, and the resultant assignment cannot be worse than the initial assignment. In our experiments, we choose *Nearest-Server Assignment* as the initial assignment for *Distributed-Greedy Assignment*.

E. Dealing With Server Capacity Constraints

So far, our proposed assignment algorithms have not assumed any capacity limitation at the servers. These "uncapacitated" algorithms are suitable for the scenario where each server site has abundant server resources or server resources can be added to these sites as needed. However, if the server capacity at each site is limited, assigning more

clients to a server than its capacity may result in significant increase in the processing delay at the server, damaging the interactivity of the DIA [19]. Therefore, we now discuss how to adapt each proposed assignment algorithm to deal with server capacity constraints.

- *Nearest-Server Assignment*: Each client first attempts to choose the nearest server as its assigned server. If the nearest server is saturated, the client in turn tries the second nearest server, the third nearest server and so on, until it finds a server which can accommodate additional clients.
- *Longest-First-Batch Assignment*: In any iteration, suppose an unassigned client c has the longest distance to its nearest server s. All unassigned clients that are nearer to server s than c are also to be assigned to s. If assigning all these clients to s overloads s, then only a portion of these clients are assigned to it to fill server s to capacity. For the remaining clients, the algorithm recomputes their nearest servers among unsaturated servers, and sorts again the distances from all unassigned clients to their nearest servers.
- *Greedy Assignment*: When selecting the pair of unassigned client and server in each iteration, the algorithm considers unsaturated servers only. After a client c is selected to be assigned to a server s in an iteration, similar to *Longest-First-Batch Assignment*, if the algorithm cannot assign to server s all clients closer to s than c due to resource constraints, it assigns only a portion of these clients to server s to fill it to capacity. Accordingly, the calculation of Δn is adjusted to reflect the capacity limitations of the servers.
- *Distributed-Greedy Assignment*: At each assignment modification, a client is allowed to be reassigned to unsaturated servers only. The "capacitated" *Nearest-Server Assignment* is used as the initial assignment.

We evaluate both "uncapacitated" and "capacitated" assignment algorithms in the next section.

V. Experimental Evaluation

We evaluate the performance of our proposed algorithms by making use of real network latency data, including the Meridian data set [1] and the MIT data set [2]. The Meridian data set contains pair-wise latency measurements between 2500 nodes in the Internet using the King measurement technique [13]. The measurements for some node pairs are not available. On discarding the nodes involved in unavailable measurements, our simulated network is represented by a complete pair-wise latency matrix for 1796 nodes. The MIT data set is a complete pair-wise latency matrix for 1024 nodes using the same measurement technique.

We assume that a client is located at each node and a certain number of servers are placed at selected nodes in the network. We simulate two types of server placements: random server placement and K-center server placement. The K-center placement is based on the minimum K-center problem that aims to place a given number of K centers in the network to minimize the maximum distance between a node and its closest center. It is widely used to model server placement in the Internet [14]. We adopt two well-known K-center algorithms for placing servers in our experiments: a 2-approximate K-center algorithm [24] and a greedy K-center heuristic [14]. We shall refer to these server placements as K-center-A and K-center-B respectively. The client assignment algorithms are evaluated with different parameter settings of server number and server capacity under random and K-center server placements.

For performance comparison, we calculate a theoretical lower bound on the maximum interaction path length as follows. The interaction path between any two clients c and c' has a length of

$$\begin{aligned} &d(c, s_A(c)) + d(s_A(c), s_A(c')) + d(s_A(c'), c') \\ &\geq \min_{s,s' \in S} \{d(c, s) + d(s, s') + d(s', c')\}. \end{aligned}$$

Thus, the maximum length of interaction paths between all client pairs has a lower bound of

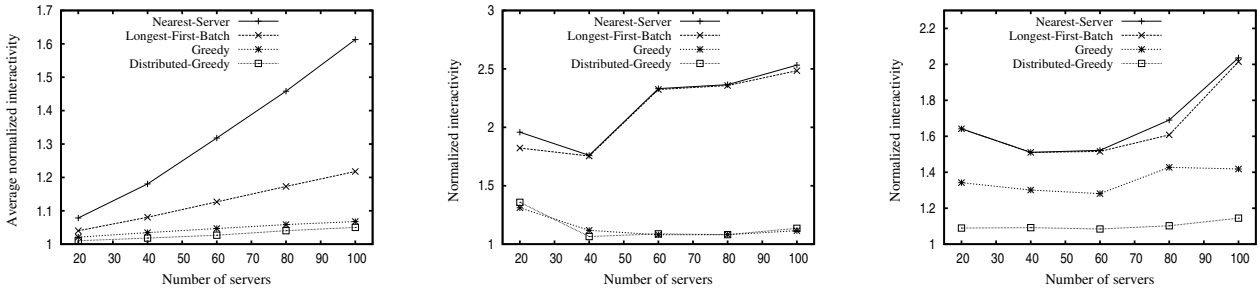
$$\max_{c,c' \in C} \min_{s,s' \in S} \{d(c, s) + d(s, s') + d(s', c')\}.$$

Note that in this lower bound, a client does not necessarily connect to a single server for interacting with all the other clients, and can choose different servers for different interactions. Thus, this lower bound is a super-optimum and may not be achievable by any real assignment. To quantify the relative performance, we normalize the maximum interaction path lengths produced by all client assignment algorithms with respect to the above lower bound. The normalized results shall be called the *normalized interactivity*.

A. Performance Comparison for Different Algorithms

First, we evaluate the performance of the client assignment algorithms without assuming any limitation on the server capacity. Fig. 7 shows the normalized interactivity of the four assignment algorithms for different server placements and numbers using the Meridian data set. The results for random server placement (Fig. 7a) are the average performance over 1000 simulation runs using 1000 different sets of randomly placed servers. The simulations using the MIT data set show similar results and are not presented here due to space limitations.

It can be seen that the two greedy assignment algorithms significantly outperform the other two assignment algorithms. In general, the performance of both greedy algorithms is close to the optimum. The interactivity resulting from *Greedy Assignment* and *Distributed-Greedy Assignment* is normally within 10% of the lower bound. Comparing the two greedy algorithms, *Distributed-Greedy Assignment* performs slightly better than *Greedy Assignment*. *Nearest-Server Assignment* produces the worst interactivity



(a) (b) (c)

Figure 7. Experimental results for different numbers of servers. (a) Random placement (b) K-center-A placement (c) K-center-B placement

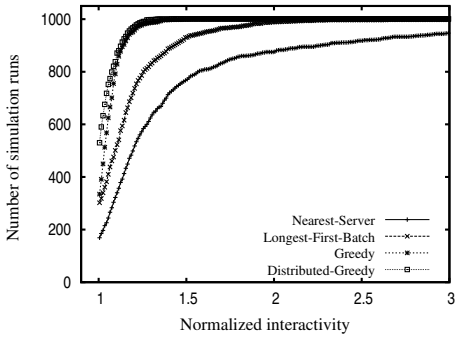


Figure 8. Cumulative distribution of the normalized interactivity for 80 servers under random server placement.

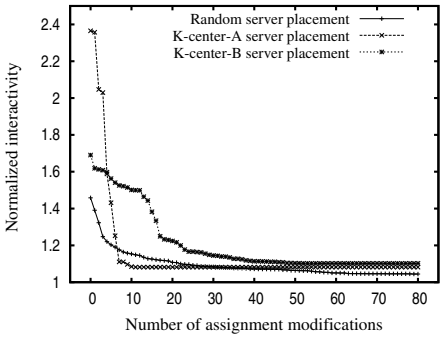


Figure 9. Performance of *Distributed-Greedy Assignment* after different numbers of assignment modifications for 80 servers.

among all algorithms. This implies that assigning each client to its nearest server is not very effective in enhancing the interactivity of continuous DIAs. Fig. 8 shows the detailed cumulative distribution of the normalized interactivity for the 1000 simulation runs of 80 randomly placed servers. As can be seen, the normalized interactivity produced by *Nearest-Server Assignment* exceeds 2 in over 100 simulation runs and exceeds 3 in over 50 runs.[2] In contrast, the other three algorithms hardly result in normalized interactivity above 2. Similar trends are also observed in the simulations with other server numbers. Compared to *Nearest-Server Assignment*, *Longest-First-Batch Assignment* improves the interactivity significantly under random server placement, but its performance is still far worse than the two greedy algorithms. Under K-center server placement, *Longest-First-Batch Assignment* performs similarly to *Nearest-Server Assignment*.

Distributed-Greedy Assignment proceeds with assignment modifications until the maximum interaction path length cannot be further reduced. To study the efficiency of *Distributed-Greedy Assignment*, we monitor its interactivity performance after each assignment modification. Fig. 9 presents the results for 80 servers under different server placements. As can be seen, the algorithm continuously improves interactivity with increasing number of assignment modifications performed and quickly converges after a few tens of assignment modifications. In general, over 99% of the total improvement is achieved after 80 assignment modifications. Note that 80 is less than 5% of the total number of clients in the network. Therefore, only a small portion of clients need to modify their assigned servers when executing *Distributed-Greedy Assignment*. Similar observations are made in the experiments for other server numbers.

B. Impact of Server Capacity

Now, we evaluate the "capacitated" version of the client assignment algorithms. We refer to the maximum number of clients that can be assigned to a server as the server capacity. Fig. 10 shows the normalized interactivity for different server capacities using the Meridian data set with 80 servers placed in the network. Again, the results for random server placement (Fig. 10a) are the average performance over 1000 simulation runs. Note that the lower bound does not change with the server capacity since its calculation assumes unlimited server capacity. When the server capacity is limited, the algorithms may not be able to assign clients

[2]The approximation ratio 3 of *Nearest-Server Assignment* does not apply to our simulations because real Internet latency data do not necessarily satisfy triangle inequality.

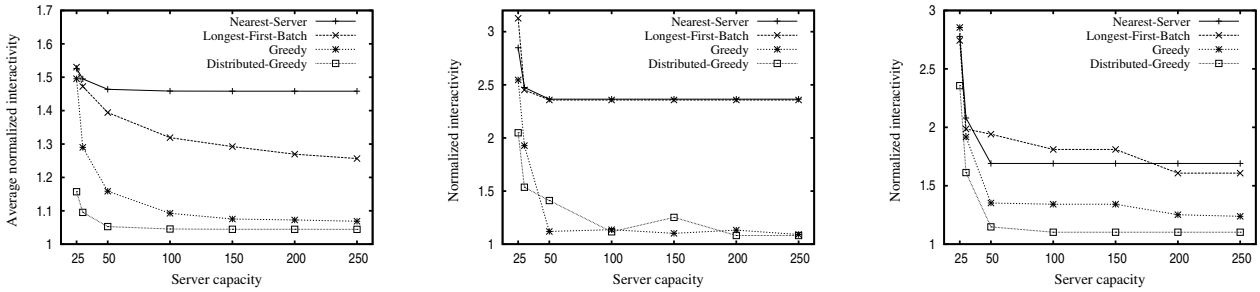


Figure 10. Experimental results for different server capacities. (a) Random placement (b) K-center-A placement (c) K-center-B placement

to desirable servers due to insufficient server capacities. Therefore, as seen from Fig. 10, the normalized interactivity of all algorithms generally gets worse with decreasing server capacity. The interactivity deteriorates more rapidly when the server capacity is severely limited. Comparing different client assignment algorithms, *Nearest-Server Assignment* and *Distributed-Greedy Assignment* are least affected by server capacity constraints. Their relative performance remains similar over different server capacities. *Distributed-Greedy Assignment* consistently and significantly improves the interactivity over *Nearest-Server Assignment*. *Longest-First-Batch Assignment* and *Greedy Assignment* are more seriously affected by server capacity constraints. This implies that their client assignments are less balanced among the servers. Under severely limited server capacities, *Longest-First-Batch Assignment* and *Greedy Assignment* produce similar or even worse interactivity than *Nearest-Server Assignment*. Overall, *Distributed-Greedy Assignment* results in the best interactivity among the four assignment algorithms.

VI. Related Work

Consistency and fairness are two important requirements for continuous DIAs. A number of synchronization mechanisms have been developed for consistency maintenance. They can be classified into optimistic and pessimistic mechanisms [9]. In optimistic mechanisms such as TimeWarp [18], [20], all servers execute operations immediately after receiving them, and the application states are repaired when inconsistency is detected. The repairs, however, may produce consistency-related artifacts that greatly disturb the user behavior. Pessimistic mechanisms such as bucket synchronization [12] and local lag [18] add artificial synchronization delays before operation execution to reduce the chance for inconsistency and artifacts to arise. For the fairness requirement, some studies used the variation of the time for each client to see the effect of its own operation as the metric of fairness [4], [6], [10], [27]. This metric, however, does not guarantee that all operations are executed in a fair order, i.e., in the order of their issuances by the clients. Some other work [16], [17] studied the required amount of delay to execute operations in the fair order at a single server. However, these results cannot be directly generalized to distributed servers. To the best of our knowledge, there has been no work on modeling the minimum achievable interaction time of continuous DIAs under the consistency and fairness requirements.

Previous studies on client assignment for interactivity enhancement considered only the client-to-server latency as the objective of optimization [22], [23], [25]. As have been shown by our results, reducing the client-to-server latency alone is not effective for improving interactivity. In an earlier work [28], we investigated client assignment for enhancing the interactivity of discrete DIAs that change their states only in response to user operations. However, no synchronization delay for preserving consistency and providing fairness was considered. Therefore, it was only applicable to discrete DIAs with great tolerance to consistency-related artifacts, such as collaborative text editors.

Another issue relevant to interactivity enhancement is server placement, which aims to find where to place servers in the network [14], [16]. Server placement is planned on long-term basis as implementing new placement solutions often involves amendment to hardware deployment or lease agreements with third-party service providers. Client assignment complements server placement in that it aims to assign clients to appropriate servers given a set of servers placed. Since client assignment deals with only software connections between clients and servers, it can be adjusted promptly to adapt to system dynamics. Our work in this paper shows that finding an optimal client assignment is a challenging task, even under carefully planned server placements.

VII. Conclusion

In this paper, we have investigated the client assignment problem for interactivity enhancement in continuous DIAs. We have modeled the interactivity performance of continuous DIAs under the consistency and fairness requirements. The minimum achievable interaction time between clients is analyzed and used as the optimization objective in our formulation of the client assignment problem. The problem

is proven to be NP-complete. Four heuristic assignment algorithms are presented and experimentally evaluated using real Internet latency data. The results show that our proposed *Distributed-Greedy Assignment* generally produces near optimal interactivity, and significantly outperforms the intuitive *Nearest-Server Assignment* algorithm and its variation.

REFERENCES

[1] The meridian latency data set. Availble from: http://www.cs.cornell.edu/People/egs/meridian/.

[2] The mit latency data set. Availble from: http://pdos.csail.mit.edu/p2psim/kingdata/.

[3] L.D. Briceño et al. Robust resource allocation in a massive multiplayer online gaming environment. In *Proc. 4th International Conference on Foundations of Digital Games*, pages 232–239, 2009.

[4] J. Brun, P. Boustead, and F. Safaei. Fairness and playability in online multiplayer games. In *Proc. of the 2nd IEEE International Workshop on Networking Issues in Multimedia Entertainment*, 2006.

[5] J. Brun, F. Safaei, and P. Boustead. Managing latency and fairness in networked games. *Communications of the ACM*, 49(11):46–51, 2006.

[6] J. Brun, F. Safaei, and P. Boustead. Server topology considerations in online games. In *Proc. 5th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2006.

[7] E. Cronin, B. Filstrup, and A. Kurc. A distributed multiplayer game server system. Technical report, University of Michigan, 2001.

[8] E. Cronin, A.R. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, 23(1):7–30, 2004.

[9] D. Delaney, T. Ward, and S. McLoone. On consistency and network latency in distributed interactive applications: A survey-Part I. *Presence: Teleoperators & Virtual Environments*, 15(2):218–234, 2006.

[10] S. Ferretti, C.E. Palazzi, M. Roccetti, M. Gerla, and G. Pau. Buscar el Levante por el Poniente: in search of fairness through interactivity in massively multiplayer online games. In *Proc. of the 2nd IEEE International Workshop on Networking Issues in Multimedia Entertainment*, 2006.

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

[12] L. Gautier, C. Diot, and J. Kurose. End-to-end transmission control mechanisms for multiparty interactive applications on the internet. In *Proc. of IEEE INFOCOM'99*, volume 3, pages 1470–1479. IEEE, 1999.

[13] K.P. Gummadi, S. Saroiu, and S.D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 5–18. ACM, 2002.

[14] S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proc. of IEEE INFOCOM'01*, volume 1, pages 31–40. IEEE, 2002.

[15] C. Jay, M. Glencross, and R. Hubbold. Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment. *ACM Transactions on Computer-Human Interaction*, 14(2), 2007.

[16] K.W. Lee, B.J. Ko, and S. Calo. Adaptive server selection for large scale interactive online games. *Computer Networks*, 49(1):84–102, 2005.

[17] Y.J. Lin, K. Guo, and S. Paul. Sync-MS: Synchronized messaging service for real-time multi-player distributed games. In *Proc. of the 10th IEEE International Conference on Network Protocols*. IEEE Computer Society, 2002.

[18] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and timewarp: Providing consistency for replicated continuous applications. *IEEE Trans. Multimedia*, 6(1):47–57, 2004.

[19] P. Morillo et al. Improving the performance of distributed virtual environment systems. *IEEE Trans. Parallel Distrib. Syst.*, 16(7):637–649, 2005.

[20] J. Müller, A. Gössling, and S. Gorlatch. On correctness of scalable multi-server state replication in online games. In *Proc. of the 5th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2006.

[21] F. Safaei et al. Latency-driven distribution: infrastructure needs of participatory entertainment applications. *IEEE Commun. Mag.*, 43(5):106–112, 2005.

[22] D.N.B. Ta and S. Zhou. A network-centric approach to enhancing the interactivity of large-scale distributed virtual environments. *Computer Communications*, 29(17):3553–3566, 2006.

[23] D.N.B. Ta and S. Zhou. A two-phase approach to interactivity enhancement for large-scale distributed virtual environments. *Computer Networks*, 51(14):4131–4152, 2007.

[24] V.V. Vazirani. *Approximation algorithms*. Springer Verlag, 2001.

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

[26] S.D. Webb, S. Soh, and W. Lau. Enhanced mirrored servers for network games. In *Proc. 6th ACM SIGCOMM workshop on Network and system support for games*, pages 117–122. ACM, 2007.

[27] S. Zander, I. Leeder, and G. Armitage. Achieving fairness in multiplayer network games through automated latency balancing. In *Proc. of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 117–124. ACM, 2005.

[28] L. Zhang and X. Tang. Client assignment for improving interactivity in distributed interactive applications. In *Proc. of IEEE INFOCOM'11*. IEEE, 2011.