The Client Assignment Problem for Continuous Distributed Interactive Applications: Analysis, Algorithms, and Evaluation

Lu Zhang and Xueyan Tang

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. Three heuristic assignment algorithms are proposed and their approximation ratios are theoretically analyzed. The performance of the algorithms is also experimentally evaluated using real Internet latency data. The experimental results show that our proposed *Greedy Assignment* and *Distributed-Modify Assignment* algorithms generally produce near optimal interactivity and significantly reduce the interaction time between clients compared to the intuitive algorithm that assigns each client to its nearest server.

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

1 Introduction

Distributed Interactive Applications (DIAs), such as multi-player online games and distributed interactive simulations, allow participants at different locations to interact with one another through networks. Thus, the interactivity of DIAs is important for participants to have enjoyable interaction experiences. Normally, interactivity is 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 [14]. We refer to this duration as the *interaction time* between participants. Network latency is known as a major barrier to provide good interactivity in DIAs [9]. It cannot be eliminated from the interactions among participants and has a lower theoretical limit imposed by the speed of light. In this paper, we focus on reducing network latency for improving interactivity in DIAs.

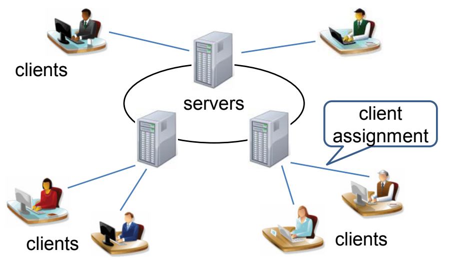


Fig. 1. Distributed server architecture.

• L. Zhang and X. Tang are with the School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore, 639798.

Increasing geographical spreads of participants in large-scale DIAs are making distributed server deployment vital for combating the network latency [1], [3]. Latency-driven distribution of servers is essential even when there are no limitations on the availability of server resources at one location [22]. 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 [4], [6]. As shown in Figure 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 due to diverse network latencies [5], [8], [11], [16], [20]. 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.

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 [20].[1] 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. Given any 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. Their approximation ratios are theoretically analyzed. The performance of the algorithms is also experimentally evaluated using real Internet latency data. The results show that our proposed *Greedy Assignment* and *Distributed-Modify Assignment* algorithms generally produce near optimal interactivity and significantly reduce the interaction time between clients compared to the intuitive *Nearest-Server Assignment* algorithm that assigns each client to its nearest server. *Distributed-Modify Assignment* also has good adaptivity to dynamics in client participation and network latency.

This paper significantly extends a preliminary conference version [26]. The rest of this paper is organized as follows. Section 2 analyzes the minimum achievable interaction time and formulates the client assignment problem. Section 3 presents the hardness results. Section 4 proposes three heuristic client assignment algorithms and analyzes their approximation ratios. The experimental setup and results are discussed in Section 5. Finally, Section 6 concludes the paper. The related work is summarized in Appendix A in the supplementary file.

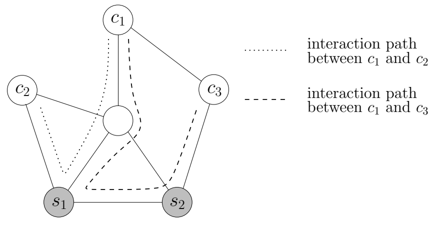
2 Problem Formulation

2.1 System Model

We model the underlying network supporting the DIA by a graph consisting of a set of nodes V. A distance $d(u, v) > 0$ is associated with each pair of nodes $(u, v) \in V \times V$, representing the network latency of the routing path 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 for sending user operations and receiving state updates. 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

1. In an earlier work [25], we have investigated client assignment for discrete DIAs that change their states in response to user operations only.



c_1
c_2
c_3
s_1
s_2
interaction path between c_1 and c_2
interaction path between c_1 and c_3

Fig. 2. Interaction paths in a network.

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 Figure 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.

The length of the interaction path between two clients c_i and c_j represents the network latency involved in their interaction, which 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 $s_A(c_i)$, which represents the network latency involved for c_i to see the effect of its own operation.

2.2 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 [20]. 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 states 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 performed by him takes effect before another action performed earlier by a different participant. 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 are of a constant lag behind the operation issuances.

2.3 Minimum Achievable Interaction Time

Given a client assignment, the minimum achievable interaction time meeting the above criterion of consistency and fairness is the maximum length of interaction paths between all client pairs, i.e.,

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

This is achievable by synchronizing the simulation times at all clients and setting the offset of each server s's simulation time relative to all clients' simulation time at

$$D - \max_{c' \in C} \{d(c', s_A(c')) + d(s_A(c'), s)\}. \qquad (1)$$

Please refer to Appendix B in the supplementary file for detailed derivations.

We present an example to illustrate the minimum achievable interaction time. In the network shown in Figure 3a, three clients c_1, c_2, c_3 are assigned to servers s_1, s_2, s_3 respectively. A straightforward strategy of time setting is to make the simulation times at all clients synchronized, and also make the simulation times at all servers synchronized (Figure 3b). The simulation time of clients must lag behind the simulation time of servers due to the network latency of delivering state updates. Suppose that at simulation time 0, one of the three clients issues an operation. If c_1 issues the operation, its operation first reaches server s_1 at simulation time 6 as shown in Figure 3b. Then, the operation is forwarded to the other two servers, which receive it at simulation time 7. Figure 3b also shows the delivery of the operation if it is issued by client c_2 or c_3. As can be seen, the latest possible simulation time for a server to receive the operation is 11 (i.e., the time for server s_2 to receive the operation if it is issued by c_3). Therefore, the constant lag for operation execution must be at least 11. If the operation is executed at simulation time 11, all the clients receive and display the resultant state update at simulation time 11, and consequently, the interaction time is 11. On the other hand, if the offset between each server and all clients is set according to (1), the delivery of operation and state update is shown in Figure 3c. It can be seen that regardless of its origin, the operation can be received and executed by all servers by simulation time 9. Thus, the resultant interaction time is 9, which equals the maximum interaction path length.

2.4 Problem Statement

The client assignment problem for maximizing the interactivity of continuous DIAs is formulated as follows:

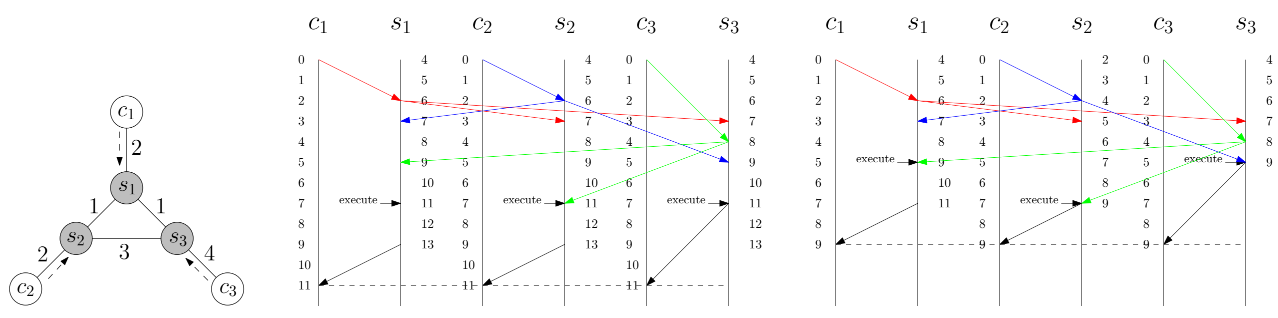
Given a set of servers S and a set of clients C in a network, find a client assignment that minimizes the maximum length of interaction paths between all client pairs, i.e., to minimize

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

2.5 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 [7]. 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 4.4 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 distance $d(u, v)$ between each node pair 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 [20] 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.



(a) Network topology (b) Interaction time when all servers are synchronized in simulation time (c) Minimum achievable interaction time

Fig. 3. An example illustrating the minimum achievable interaction time.

3 Hardness Results

Theorems 1 and 2 below present the hardness results of the client assignment problem. Please refer to Appendix C in the supplementary file for the detailed proofs.

Theorem 1: The client assignment problem is NP-complete.

Theorem 2: No polynomial-time algorithm for the client assignment problem can achieve an approximation ratio less than $4/3$ if the network latency satisfies the triangle inequality and any bounded approximation ratio otherwise, if P$\neq$NP.

4 Heuristic Algorithms

In this section, we present three heuristic client assignment algorithms. The computation of these algorithms is based on the network latencies between clients and servers, which can be obtained with existing tools like ping and King [12].

4.1 Nearest-Server Assignment

The first algorithm is called *Nearest-Server Assignment*, which intuitively assigns clients to their nearest servers [16], [23]. 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 $O(|S|)$. While this *Nearest-Server 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.

The assumption of the triangle inequality is commonly made when theoretically analyzing the performance of the approximation algorithms in facility location problems [15].[2] When assuming that the network latency satisfies the triangle inequality, we can show that *Nearest-Server Assignment* has a tight approximation ratio of 3. In the absence of the triangle inequality [18], *Nearest-Server Assignment* cannot achieve any bounded approximation ratio. Please refer to Appendix D in the supplementary file for the detailed proof.

Theorem 3: Nearest-Server Assignment has an approximation ratio of 3 for networks with the triangle inequality.

2. We count the number of triples (u, v, w) that satisfy the constraint $d(u, v) + d(u, w) \geq d(v, w)$ in the Internet latency data set used in our experimental evaluation (Section 5) and observe that 92.6% of the triples satisfy the triangle inequality.

4.2 Greedy Assignment

The second algorithm *Greedy Assignment* adopts a greedy approach to assign clients iteratively, starting with an empty assignment. In each step, the algorithm considers all the possibilities of assigning an unassigned client to a server. If a client c is selected to be assigned to a server s, then all unassigned clients that are not farther from s than c are also assigned to s as this would not increase the maximum interaction path length. Let Δn be the number of new clients assigned to s in this way, and Δl be the increase in the maximum interaction path length due to these new assignments. To minimize the amortized increase in the maximum interaction path length, we use $\Delta l/\Delta n$ as the cost metric for selecting which client to be assigned to which server. In each step, among all possible pairs of unassigned client and server, the pair (c, s) resulting in the minimum cost $\Delta l/\Delta n$ is selected and the corresponding clients are then 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 are sorted in a list L_s in a preprocessing stage. This sorted list is then incrementally updated by removing newly assigned clients at the end of every step. As a result, 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 maximum interaction path length before assigning c with the maximum length of the interaction paths from c to all the clients already assigned. The latter is given by

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

where $2d(c, s)$ is the interaction path length from c to itself and C' is the set of clients already assigned. 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 Algorithm 1. *Greedy Assignment* is suited for centralized implementation due to its need for global knowledge about the distances between clients and servers.

In the preprocessing stage, sorting lists L_s for all servers and calculating the indexes of all clients in the lists can be done in $O(|S||C|\log |C|)$ time. Suppose all clients are assigned to

Algorithm 1: The *Greedy Assignment* algorithm

```
1  $C' \leftarrow \emptyset$; // the set of clients already assigned
2  $max\_len \leftarrow 0$; // the maximum interaction path length
3  forall $s \in S$ 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 $i \leftarrow 1$ to $|C|$ do
7          $index[s, L_s[i]] \leftarrow i$;
8  while $C' \neq C$ do
9      $min \leftarrow \infty$;
10     forall $s \in S$ do
11         $m \leftarrow \max_{b \in C'}\{d(s, s_A(b)) + d(s_A(b), b)\}$;
12         forall $c \in C - C'$ 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     forall $c \in C - C'$ such that $d(c, s^*) \leq d(c^*, s^*)$ do
24         set $s_A(c) = s^*$;
25         $C' \leftarrow C' \cup \{c\}$;
26     forall $s \in S$ do
27         $nuc \leftarrow 0$; // the number of unassigned clients
28         for $i \leftarrow 1$ to $|C|$ do
29             if $L_s[i] \in C - C'$ then
30                 $nuc \leftarrow nuc + 1$;
31             $index[s, L_s[i]] \leftarrow nuc$;
```

servers in m steps. To calculate the time complexity of each step, 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, the 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 step 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 step 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 this stage in each step is $O(|S||C|)$. So, the total time complexity for one step 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 $m \leq |C|$.

Theorem 4 presents the approximation ratio of *Greedy Assignment* that is asymptotically tight, when assuming that the network latency satisfies the triangle inequality. In the absence of the triangle inequality, *Greedy Assignment* has unbounded approximation ratio. Please refer to Appendix E in the supplementary file for the detailed proof.

Theorem 4: Greedy Assignment has an approximation ratio of $O(\log(n))$ for networks with the triangle inequality, where n is the number of clients.

4.3 Distributed-Modify Assignment

The third algorithm *Distributed-Modify Assignment* is performed in a distributed manner without requiring the global knowledge of the network at any single server. It starts with an initial assignment. Then, the assignment is continuously modified for reducing the maximum interaction path length D until it cannot be further reduced. This process is referred to as the assignment modification.

One server is elected as a coordinator responsible for calculating D and selecting the server to perform the assignment modification. To calculate D of the initial assignment, each server measures its distances (network latencies) to all the other servers. It also measures its distances to all the clients that are assigned to it and maintain them as a sorted list. Then, each server s broadcasts to all the other servers its longest distance $l(s)$ to its clients, and sends the inter-server distances to the coordinator. The coordinator calculates D based on the received information.

To perform the assignment modification, the coordinator selects a server s that is involved in a longest interaction path and informs s of the current D value. Then, server s attempts to reduce D by modifying the assignment of its client c that introduces the 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 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 $L(s^*)$. In this case, servers s and s^* update their longest distances to their clients and broadcast these distances to all the other servers if they are changed. Finally, the coordinator recalculates D. If D cannot be reduced after the coordinator has tried all the servers involved in the longest interaction path(s), the algorithm terminates.

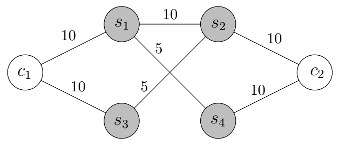


Fig. 4. An example in which changing the assigned servers of two clients simultaneously increases the maximum interaction path length.

If the coordinator selects two or more servers to perform assignment modifications 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. Figure 4 gives an example. Suppose that clients c_1 and c_2 are initially assigned to servers s_1 and s_2 respectively, so that the maximum interaction path length is 30. If c_1 (or c_2) changes its assigned server to s_3 (or s_4), the maximum interaction path length would be reduced to 25. However, if both clients change their assigned servers, the interaction path length between c_1 and c_2 would become 40, which is even longer than the maximum interaction path length of the initial assignment. Therefore, the coordinator selects only one server at a time for assignment modification.

In each assignment modification, the computation of $L(s')$ by each server s' has a time complexity of $O(|S|)$. The time complexity for server s to find s^* with the minimum $L(s^*)$ is $O(|S|)$. If client c is reassigned, its new server s^* has an additional time complexity of $O(\log|C|)$ for updating the sorted list of the distances to its clients. The time complexity for the coordinator to recalculate D is $O(|S|^2)$. In the process of an assignment modification, s broadcasts to all the other servers the information of c and then they reply to s with $L(s')$. In addition, c's new server s^* and server s also broadcast the new $l(s^*)$ and $l(s)$ to all the other servers. Therefore, the message complexity of one assignment modification is $O(|S|)$.

Distributed-Modify Assignment has unbounded approximation ratio if it starts with an arbitrary initial assignment, even for networks with the triangle inequality. Please refer to Appendix F in the supplementary file for details. On the other hand, if *Distributed-Modify Assignment* takes *Nearest-Server Assignment* as the initial assignment, the resultant assignment cannot be worse than the latter since the assignment modification can only reduce the maximum interaction path length. Our experiments in Section 5 show that *Distributed-Modify Assignment* normally outperforms *Nearest-Server Assignment* significantly.

4.4 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 [19]. 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. Therefore, we now discuss how to adapt each proposed assignment algorithm to deal with server capacity constraints.

- *Nearest-Server Assignment*: Each client chooses its server and makes the request to connect to the server independently. Each server accepts the client requests on a first-come-first-serve basis until it is saturated. A client first attempts to choose the nearest server. If the nearest server is saturated, the client in turn tries the second nearest server, the third nearest server and so on, until its connection request is accepted by a server.
- *Greedy Assignment*: When selecting the pair of unassigned client and server in each step, the algorithm considers unsaturated servers only. After a client c is selected to be assigned to a server s in a step, if the algorithm cannot assign to server s all clients closer to s than c due to the capacity constraint of s, only a portion of these clients are assigned to server s to fill it to capacity. Accordingly, the calculation of Δn is adjusted to reflect the capacity limitations of the servers.
- *Distributed-Modify Assignment*: At each assignment modification, a client is allowed to be reassigned to unsaturated servers only.

The approximation ratios previously analyzed for "uncapacitated" assignment algorithms are not applicable to "capacitated" assignment algorithms. *Distributed-Modify Assignment* has unbounded approximation ratio even without server capacity limitation. Thus, the same is also true when the server capacity is limited. Theorems 5 and 6 below present the tight approximation ratios of the "capacitated" *Nearest-Server Assignment* and the "capacitated" *Greedy Assignment* respectively, when assuming that the network latency satisfies the triangle inequality. Please refer to Appendices G and H in the supplementary file for the detailed proofs.

Theorem 5: The "capacitated" *Nearest-Server Assignment* has an approximation ratio of 5 for networks with the triangle inequality.

Theorem 6: The "capacitated" *Greedy Assignment* has an approximation ratio of $O(n)$ for networks with the triangle inequality, where n is the number of clients.

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

5 Experimental Evaluation

We evaluate the performance of the proposed algorithms by simulation experiments making use of the Meridian data set [24]. The data set contains pair-wise latency measurements between $2,500$ nodes in the Internet using the King measurement technique. The measurements of some node pairs are missing from the data set. On discarding the measurements that involve these nodes, the network that we simulate is represented by a complete matrix of pair-wise latencies among $1,796$ nodes. One client is assumed to be located at each node, and a certain number of servers are placed at selected nodes in the network. The proposed assignment algorithms are evaluated with various parameter settings of server number and capacity. In the default setting, there is no constraint on the server capacity. We use *Nearest-Server Assignment* as the initial assignment of *Distributed-Modify Assignment* in our experiments.

We simulate three types of server placement: random, K-center, and K-median. In random server placement, for each parameter setting, we execute $1,000$ simulation runs using $1,000$ different sets of randomly placed servers. Unless stated otherwise, the average performance of these $1,000$ simulation runs is plotted together with the 90th and 10th percentile results. K-center server placement is based on the K-center problem which aims to place k servers (centers) in the network so that the maximum distance from the clients (nodes) to their nearest servers is minimized. K-median server placement is based on the K-median problem which intends to minimize the average distance from all clients to their nearest servers. These two problems are widely used to model server placement in the Internet [7]. Since they are both NP-complete [10], we adopt a greedy K-center algorithm [7] and a greedy K-median algorithm [21] for placing servers in the experiments.

We develop a theoretical lower bound on the maximum interaction path length to quantify the relative performance of the assignment algorithms. The interaction path between any two clients c and c' has a length of

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

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

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

We normalize the maximum interaction path lengths produced by all assignment algorithms with respect to the above lower bound. The normalized results shall be called the *normalized interactivity*. Note that the above lower bound calculation does not enforce that each client is assigned to a single server through which it interacts with all the other clients. Thus, this lower bound is a super-optimum that may not be achievable by any real assignment.

5.1 Comparison of Proposed Algorithms

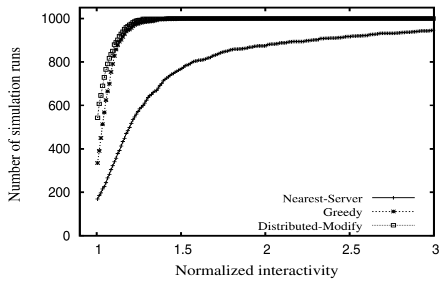


Fig. 6. Cumulative distribution of the normalized interactivity under 80 randomly placed servers.

First, we compare the performance of the client assignment algorithms without assuming any limitation on the server capacity. Figure 5 shows the normalized interactivity of each algorithm as a function of the number of servers in the network. As seen from Figure 5a, the average performance of *Nearest-Server Assignment* under random server placement is significantly worse than that of the other two algorithms for all server numbers tested. Figure 6 shows the detailed cumulative distribution of the normalized interactivity for the $1,000$ simulation runs of 80 randomly placed servers. The normalized interactivity produced by *Nearest-Server Assignment* exceeds 2 in over 100 simulation runs and exceeds 3 in over 50 runs. In contrast, the other two algorithms hardly result in any normalized interactivity above 2. Under K-center and K-median server placements, *Nearest-Server Assignment* also performs far worse than the other two heuristics as shown in Figures 5b and 5c. The above results imply that intuitively assigning each client to its nearest server is not the most effective way of enhancing interactivity, even under carefully planned server placements. Comparing *Greedy Assignment* and *Distributed-Modify Assignment*, their performance is generally close to each other. The interactivity produced by both algorithms is within normally 40% of the lower bound, which means that it is within at most the same percentage of the optimum.

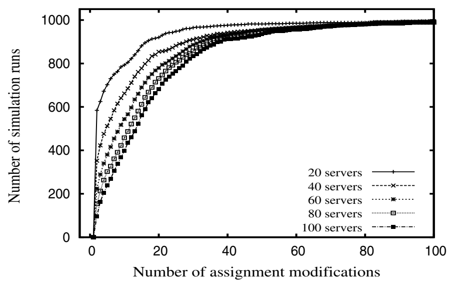


Fig. 7. Cumulative distribution of the number of assignment modifications performed by *Distributed-Modify Assignment*.

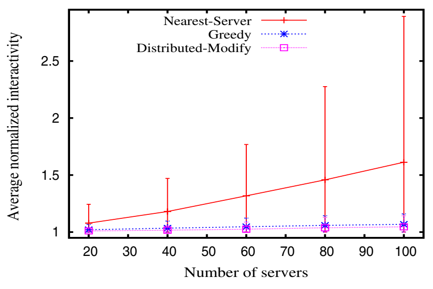
TABLE 1
Computation times for networks with 80 servers

Algorithm	CPU Time		
	Random	K-center	K-median
Nearest-Server	0.79s	0.80s	0.81s
Greedy	3.26s	2.97s	2.22s
Distributed-Modify	1.42s	1.19s	1.20s

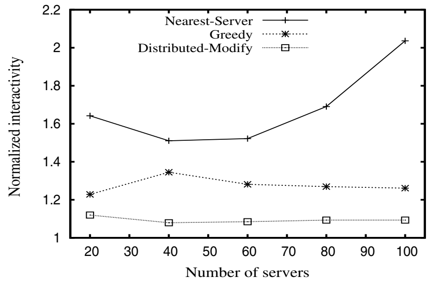
Note that *Distributed-Modify Assignment* proceeds with assignment modifications until the maximum interaction path length cannot be further reduced. Figure 7 shows the distribution of the numbers of assignment modifications performed in the simulation. We count each attempt made to change the assigned server of a client involved in a longest interaction path as one assignment modification, even if it is decided not to change the assigned server of the client after calculation. As seen from Figure 7, the algorithm performs fewer than 50 assignment modifications in 94% of the simulation runs. In fact, the average number of assignment modifications performed is less than 18. Under K-center and K-median placements, the numbers of assignment modifications are about 55 and 43 on average. These results imply that only a small proportion of clients need to modify their servers in the execution of *Distributed-Modify Assignment*.

Table 1 reports the computation times of the algorithms under 80 servers of various placements in our simulation performed on a PC with a Pentium Xeon 3.2 GHz CPU and 6 GB RAM. It can be seen that *Distributed-Modify Assignment* has much shorter computation time than *Greedy Assignment*.

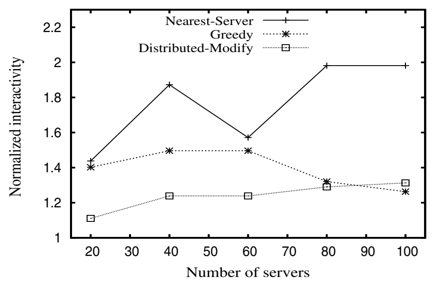
Next, we evaluate the "capacitated" assignment algorithms under various server capacity constraints. We refer to the maximum number of clients that is allowed to be assigned to a server as the server capacity. All servers are assumed to have the same capacity. The "capacitated" *Nearest-Server Assignment* is used as the initial assignment of the "capacitated" *Distributed-Modify Assignment*. Figure 8 shows the normalized interactivity for different server capacities when 80 servers are placed in the network. Note that the theoretical lower bound does not change with the server capacity since its calculation assumes unlimited server capacity. As expected, the interactivity performance produced by all algorithms deteriorates as the server capacity decreases. This is because when the server capacity is limited, the algorithms may not be able to assign clients to their most preferred servers. Com-



(a) Random placement

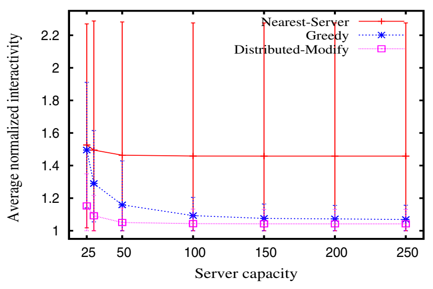


(b) K-center placement

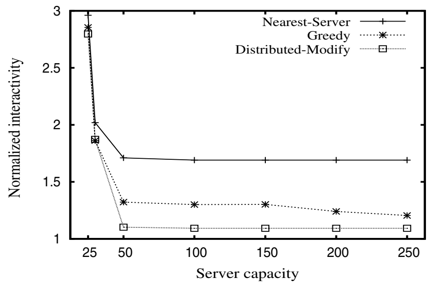


(c) K-median placement

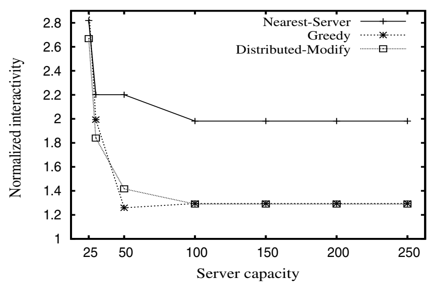
Fig. 5. Experimental results for different numbers of servers.



(a) Random placement



(b) K-center placement



(c) K-median placement

Fig. 8. Experimental results for different server capacities.

paring different algorithms, *Nearest-Server Assignment* and *Distributed-Modify Assignment* are generally less affected by server capacity constraints than *Greedy Assignment*. This implies that the client assignments of *Nearest-Server Assignment* and *Distributed-Modify Assignment* are more balanced among servers. The relative performance of the three assignment algorithms remains similar over different server capacities unless the server capacity is severely limited. When the server capacity is severely limited, all the algorithms produce similar interactivity because there is little space for optimizing client assignment to improve interactivity. Similar results are also observed for other server numbers, which are not shown here due to space limitations.

5.2 Dynamic Scenarios

So far, we have evaluated the interactivity performance of the proposed algorithms in static scenarios where the client participation and network latency do not change. In practice, DIAs often support dynamic client participation such that clients may join and leave the network at any time. In addition, network latency may vary with time depending on network conditions. Previous experiments have shown that *Greedy Assignment* performs similarly to *Distributed-Modify Assignment* and outperforms *Nearest-Server Assignment* in static scenarios. However, it is a centralized algorithm and may not be efficient in dealing with dynamic scenarios. On the other hand, *Distributed-Modify Assignment* is more suitable for adapting the client assignment to dynamics in client participation and network latency due to its distributed and incremental nature. In this section, we evaluate the adaptivity of *Distributed-Modify Assignment* in dynamic scenarios. We also involve the other two assignment algorithms in the experiments for comparison purpose.

5.2.1 Adaptivity to Client Joining/Leaving

We first consider the scenario in which clients can join and leave the network freely. To model the joining and leaving patterns of clients, we randomly assign each client two weights w_j and w_l ($0 < w_j, w_l < 1$). w_l indicates the chance of the client leaving the DIA when it is currently participating, and w_j indicates the chance of the client joining the DIA when it is not participating at present. Small w_j and w_l mean that the client is inclined to keep its current state of participation unchanged, while large w_j and w_l imply that the client tends to change its participation state frequently. The client participation behavior is then modeled by a sequence of events, each of which could be either a join event or a leave event. Each event is randomly generated based on the weights and participation states of all clients. We start from a network in which no client is participating and simulate a sequence of $10,000$ events. It is observed that about half of the clients participate in the stable state. Please refer to Appendix I in the supplementary file for the evolution of the number of participating clients.

To adapt to dynamic client participation, *Nearest-Server Assignment* simply assigns each joining client to its nearest server. For *Distributed-Modify Assignment*, whenever a client joins the network, it is initially assigned to its nearest server. Then, for each join and leave event, based on the existing assignment, assignment modifications are performed until the maximum interaction path length cannot be further reduced. For *Greedy Assignment*, we record the clients assigned at each step and the cost of assigning them. When a participating client c leaves, the clients that are assigned before c in the previous execution are directly assigned to the same servers as in the previous execution. Then, the original *Greedy Assignment* is executed to assign the remaining clients. When a new client c joins, assume that in *Greedy Assignment*, clients are assigned in the same way as in the previous execution up to step $i-1$. Let l_i be the maximum interaction path length and n_i be the number of unassigned clients before step i starts. Then, if client c is assigned at step i, the increase in the maximum interaction path length cannot be less than $2 \cdot \min_{s \in S} d(c,s) - l_i$, and the number of clients assigned at this step cannot be larger than n_i. Thus, the cost of assigning c at step i cannot be less than $x_i = \big(2 \cdot \min_{s \in S} d(c,s) - l_i\big)/n_i$. If x_i is larger than the cost of step i of the previous execution, client c cannot be assigned at step i, which implies that the newly assigned clients at this step would be the same as in the previous execution. Therefore, instead of executing the original *Greedy Assignment* algorithm, we can assign clients in the same way as in the previous execution as long as x_i is larger than the cost of step i of the previous execution. After encountering the first x_i not exceeding the cost of step i, we start executing the original *Greedy Assignment* algorithm to assign the remaining clients.

Figure 9 shows the interactivity performance of each algorithm when 80 servers are placed in the network. The results for random server placement are those for one sample placement. As can be seen, the relative performance of all algorithms remains similar to the results in the static scenario. In general, *Distributed-Modify Assignment* consistently achieves the best interactivity among the three algorithms. Our statistics show that the average numbers of assignment modifications performed by *Distributed-Modify Assignment* at each join and leave event are just 1.95, 1.57 and 1.87 under the three server placements. This implies that performing just a few assignment modifications per event is sufficient to maintain decent interactivity performance.

5.2.2 Adaptivity to Dynamics in Network Latency

The latency for any information to be transmitted across the network is affected by the network conditions and may vary with time. In this section, we evaluate the adaptivity of the algorithms to the variance of network latency. Since the Meridian data set does not contain latency measurements at different times, we employ the PlanetLab All-Pairs-Pings data set [2] which includes continuous pair-wise measurements of the latency between PlanetLab nodes using ping. The data we use for simulation is collected within a one-day period (May/9/2005) during which there are 188 PlanetLab nodes consistently involved in all the measurements. One measurement is performed every 15 minutes and consequently there are 95 sets of measurements. Thus, we simulate a network consisting of 188 nodes and emulate the dynamics in network latency using these 95 sets of latency measurements. Again, a client is assumed to be located at each node.

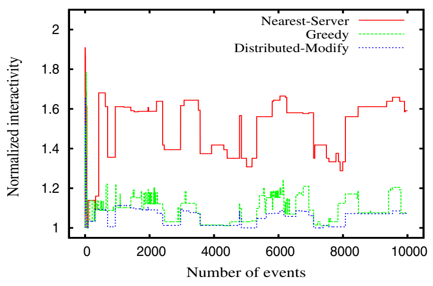
Similar methods are employed to deal with dynamic network latency. After each latency measurement, *Nearest-Server Assignment* simply allows each client to update its nearest server. For *Distributed-Modify Assignment*, assignment modifications are performed to adjust the existing assignment. *Greedy Assignment* is re-executed from scratch.

Figure 10 presents the normalized interactivity of all algorithms over the day with 15 servers placed in the network. The K-center and K-median placements are calculated based on the first set of latency measurements. This models the scenario in which the network latency information used for the deployment of servers is predicted by the first measurement. The theoretical lower bound on the maximum interaction path length is recomputed after each set of measurements for deriving the normalized interactivity. It can be seen from Figure 10 that *Distributed-Modify Assignment* produces better interactivity and performs much more stably than *Nearest-Server Assignment* under all server placements. The average numbers of assignment modifications performed after each latency measurement are just 5.73, 9.02 and 5.95 under the three server placements. *Greedy Assignment* generally produces similar interactivity performance to *Distributed-Modify Assignment*. However, as stated earlier, *Greedy Assignment* is a centralized algorithm which would introduce much higher computation and communication overheads than *Distributed-Modify Assignment* in adapting the client assignment.

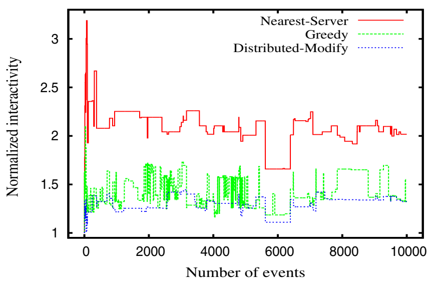
6 Conclusion

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. Three heuristic assignment algorithms are presented. Their approximation ratios are theoretically analyzed and their performance is experimentally evaluated using real Internet latency data. The results show that: (1) our proposed *Greedy Assignment* and *Distributed-Modify Assignment* algorithms significantly outperform the intuitive *Nearest-Server Assignment* algorithm; (2) *Distributed-Modify Assignment* requires only a small proportion of clients to perform assignment modifications for improving interactivity; and (3) *Distributed-Modify Assignment* has good adaptivity to dynamics in both client participation and network latency.

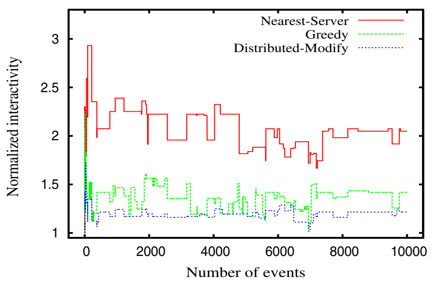
In this paper, we have assumed that $d(u,v) = d(v,u)$ for any pair of nodes u and v. To deal with asymmetric routing [13], the network can be modeled by a directed graph. Each pair of nodes is associated with the lengths of two routing paths of different directions. The interaction path from a client c_i to another client c_j can be considered as a directed path that is different from the interaction path from c_j to c_i. It is easy to show that if we change the definition of D to be the maximum length of all the directed interaction paths between clients, the consistency and fairness requirements can still be satisfied. Therefore, the objective of the client assignment problem becomes to minimize the maximum length of all directed interaction paths. For the heuristic algorithms, we can simply use the lengths of the directed routing paths between



(a) One sample random placement

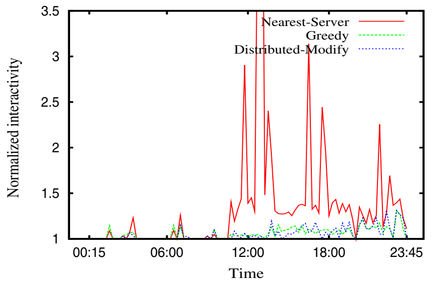


(b) K-center placement

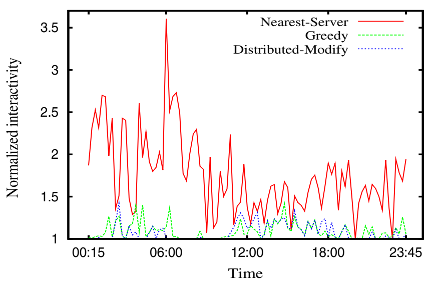


(c) K-median placement

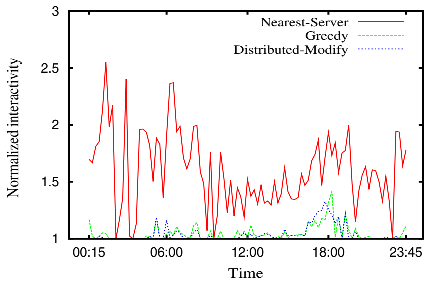
Fig. 9. Performance of client assignment algorithms with dynamic client participation.



(a) One sample random placement



(b) K-center placement at initial latency measurement



(c) K-median placement at initial latency measurement

Fig. 10. Performance of client assignment algorithms with dynamic network latency.

clients and servers in the calculation without modifying the algorithms. However, the approximation ratios of the algorithms may change. We leave the detailed analysis to the future work.

References

[1] LotRO server list. http://lotro-wiki.com/index.php/List_of_Worlds.

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

[3] WoW server list. http://www.wowwiki.com/Realms_list.

[4] L.D. Briceño, H.J. Siegel, A.A. Maciejewski, Y. Hong, B. Lock, M.N. 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*, pages 232–239, 2009.

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

[7] E. Cronin, S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. *IEEE J. Sel. Areas Commun.*, 20(7):1369–1382, 2002.

[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] 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.

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

[12] 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 Measurement*, pages 5–18, 2002.

[13] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker. On routing asymmetry in the Internet. In *Proc. IEEE GLOBECOM'05*, 2005.

[14] 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.

[15] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000.

[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. IEEE ICNP'02*, 2002.

[18] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee. Triangle inequality and routing policy violations in the Internet. In *Proc. PAM 2009*, pages 45–54, 2009.

[19] M. Marzolla, S. Ferretti, and G. D'Angelo. Dynamic resource provisioning for cloud-based gaming infrastructures. *ACM Computers in Entertainment*. To appear.

[20] 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.

[21] L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *Proc. IEEE INFOCOM'01*, pages 1587–1596.

[22] F. Safaei, P. Boustead, C.D. Nguyen, J. Brun, and M. Dowlatshahi. Latency-driven distribution: infrastructure needs of participatory entertainment applications. *IEEE Commun. Mag.*, 43(5):106–112, 2005.

[23] 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, 2007.

[24] B. Wong, A. Slivkins, and E.G. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *ACM SIGCOMM'05*, pages 85–96, 2005.

[25] L. Zhang and X. Tang. Client assignment for improving interactivity in distributed interactive applications. In *Proc. IEEE INFOCOM'11*, pages 3227–3235, 2011. (An extended version is accepted to appear in IEEE/ACM Transactions on Networking.).

[26] L. Zhang and X. Tang. The client assignment problem for continuous distributed interactive applications. In *Proc. IEEE ICDCS'11*, pages 203–214, 2011.

Lu Zhang 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. He is a student member of the IEEE.

Xueyan Tang 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, and wireless sensor networks. He is a senior member of the IEEE.