Client Assignment for Improving Interactivity in Distributed Interactive Applications

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

***Abstract*—Distributed Interactive Applications (DIAs) are networked systems that allow multiple participants to interact with one another in real time. Wide spreads of client locations in large-scale DIAs often require geographical distribution of servers to meet the latency requirements of the applications. In the distributed server architecture, how the clients are assigned to the servers directly affects the network latency involved in the interactions between clients. This paper focuses on the client assignment problem for enhancing the interactivity performance of DIAs. We formulate the problem as a combinational optimization problem on graphs and prove that it is NP-complete. Several heuristic algorithms are proposed for fast computation of good client assignments and are experimentally evaluated. The experimental results show that the proposed greedy algorithms perform close to the optimal assignment and generally outperform the *Nearest-Assignment* algorithm that assigns each client to its nearest server.**

I. Introduction

Distributed Interactive Applications (DIAs) are networked systems that allow multiple participants at different locations to interact with each other in real time through their computers. DIAs spread over a wide range of areas that are gaining popularity rapidly, such as multiplayer online games, distributed interactive simulations, and collaborative computer-aided design and engineering. Typically, the application's state (such as the virtual worlds in multi-player online games and the shared workspaces in collaborative design tools) is maintained by servers. Participants, known as clients, are responsible for sending user-initiated operations to the servers and receiving updates of the application's state from the servers. Since DIAs are human-in-the-loop applications, it is of crucial importance to improve the interactivity of DIAs for supporting graceful interactions among clients. The major challenges to interactivity enhancement are to deal with network latencies. Wide spreads of client locations in large-scale DIAs often require geographical distribution of servers to meet the latency requirements of the applications. This kind of latency-driven distribution is essential even when there are no limitations on the availability of server resources in one location [9].

In the distributed server architecture, the servers communicate with each other directly, while each client is assigned to one server and interacts with other clients through their assigned servers [1], [4]. We characterize the interactivity performance of DIAs by the duration from the time when a client issue an operation to the time when the resultant state update is presented to the same client or a different client. This duration shall be called the *interaction time* between these clients. Under the distributed server architecture, the interaction time between any pair of clients includes the network latencies between the clients and their assigned servers and the network latency between their assigned servers. These network latencies are directly affected by how the clients are assigned to the servers. Therefore, effectively assigning clients to appropriate servers in the DIA is an important issue for enhancing its interactivity. We refer to this issue as the *client assignment problem*.

Finding the optimal client assignment is a challenging task. An intuitive approach for reducing the latency from the clients to their assigned servers is to assign each client to its nearest server [7], [14]. This assignment, however, may considerably increase the latency between the assigned servers of different clients and thus perform far worse than optimum, as shall be shown by our experimental results. On the other hand, assigning all clients to a single server eliminates the contribution of the inter-server latency to the interaction time, but such assignment may significantly increase the latency between the clients and their assigned server, defeating the purpose of geographical distribution of servers. An optimal assignment for maximizing interactivity performance should strike a balance between the inter-server latency and the client-to-server latency.

In this paper, we investigate the client assignment problem for enhancing the interactivity performance of DIAs. We formulate the problem as a combinational optimization problem on graphs and prove that it is NP-complete. Several heuristic assignment algorithms are proposed and experimentally evaluated. Simulations using real Internet latency data show that the proposed algorithms are efficient and effective in reducing the interaction time between clients.

Some existing work on how to assign clients to servers in distributed virtual environments partitions the clients into groups and assign each client group to one server [10], [11]. Only the clients in the same group are allowed to interact with each other. As a result, the objective of the assignment is simply to reduce the client-server latencies. Similarly, Webb *et al.* [13] considered the client-server latencies only in studying the client assignment for online games. Different from existing research, we integrate the client-server and inter-

server latencies in the formulation of interaction time to model interactions between any clients in the system. Lee *et al.* [7] investigated where to place servers in the network for satisfying a given latency bound between clients. In contrast, we aim to assign clients to appropriate servers for maximizing interactivity given a set of servers placed in the network.

The rest of this paper is organized as follows. Section II presents the system model and a formulation of the client assignment problem. Section III analyzes the characteristics of an optimal client assignment and proves the NP-completeness results. Section IV proposes three heuristic assignment algorithms for improving the interactivity of DIAs. The experimental setup and results are discussed in Section V. Finally, Section VI concludes the paper.

II. Problem Formulation

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

We assume a distributed server architecture. Let $C \subseteq V$ be the set of clients in the network and $S \subseteq V$ be the set of servers in the network. Each client should be assigned to a server in order to send operations and receive state updates. An assignment A is a mapping from C to S, where for each client $c \in C$, we denote by $s_A(c) \in S$ as the server that client c is assigned to. In this paper, we focus on reducing the network latency involved in the interaction between clients, since the network latency is more difficult to improve than the processing delay at the servers [3]. A busy server can always be better provisioned to meet the capacity requirements, e.g., by forming a server cluster. Nevertheless, we do discuss how to deal with the constraint of limited server capacities in our proposed assignment algorithms in Section IV-D.

For two clients c_i and c_j to interact, the communication should go through their assigned servers. To be specific, if c_i issues an operation, the following steps have to be taken in order for c_j to see the effect of the operation. 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 and 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 are involved in the interaction process for c_i to see the effect of the operation. Therefore, we refer to the concatenation of these three paths as the *interaction path* between c_i and c_j. The length of the interaction path, denoted as $d_A(c_i, c_j) = d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)$, represents the interaction time between c_i and c_j in assignment A. If c_i and c_j are assigned to the same server, then the length of their interaction path is simply $d_A(c_i, c_j) = d(c_i, s_A(c_i)) + d(s_A(c_j), c_j)$. If c_i and c_j are the same client, the length of their interaction path $d_A(c_i, c_i) = 2 \cdot d(c_i, s_A(c_i))$ is the round-trip time between c_i and its assigned server $s_A(c_i)$, and represents the interaction time for client c_i to see the effect of its own operation.

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

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

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

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

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

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

III. NP-Completeness Results

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

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

Proof: Suppose A is an assignment in which c is assigned to s_2. Based on assignment A, we construct a new assignment A' by changing the assigned server of c from s_2 to s_1, and keeping the assigned servers of other clients unchanged, i.e.,

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

$s_{A'}(c) = s$, and $\forall c_j \neq c,\ s_{A'}(c_j) = s_A(c_j)$. Then, for any clients $c_i, c_j \neq c$, we have $d_{A'}(c_i, c_j) = d_A(c_i, c_j)$. Thus, the length of interaction paths in assignment A' is given by

$$\begin{aligned}
D(A') &= \sum_{c_i,c_j \in C} d_{A'}(c_i, c_j) \\
&= \sum_{c_j \in C} d_{A'}(c, c_j) + \sum_{c_i,c_j \neq c} d_{A'}(c_i, c_j) \\
&= \sum_{c_j \neq c} \big(d(c, s_1) + d(s_1, s_{A'}(c_j)) + d(s_{A'}(c_j), c_j)\big) \\
&\quad + 2d(c, s_1) + \sum_{c_i,c_j \neq c} d_{A'}(c_i, c_j) \\
&= \sum_{c_j \neq c} \big(d(c, s_1) + d(s_1, s_A(c_j)) + d(s_A(c_j), c_j)\big) \\
&\quad + 2d(c, s_1) + \sum_{c_i,c_j \neq c} d_A(c_i, c_j).
\end{aligned}$$

By the triangle inequality, we have $d(s_1, s_A(c_j)) \leq d(s_1, s_2) + d(s_2, s_A(c_j))$. Therefore,

$$\begin{aligned}
D(A') \leq \sum_{c_j \neq c} &\big(d(c, s_1) + d(s_1, s_2) + d(s_2, s_A(c_j)) \\
&+ d(s_A(c_j), c_j)\big) + 2d(c, s_1) + \sum_{c_i,c_j \neq c} d_A(c_i, c_j).
\end{aligned}$$

Since $d(c, s_1) < d(c, s_1) + d(s_1, s_2) = d(c, s_2)$, it follows that

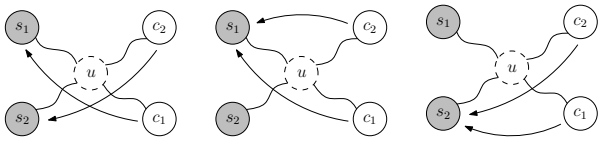
$$\begin{aligned}
D(A') &< \sum_{c_j \neq c} \big(d(c, s_2) + d(s_2, s_A(c_j)) + d(s_A(c_j), c_j)\big) \\
&\quad + 2d(c, s_2) + \sum_{c_i,c_j \neq c} d_A(c_i, c_j) \\
&= \sum_{c_j \in C} d_A(c, c_j) + \sum_{c_i,c_j \neq c} d_A(c_i, c_j) = D(A).
\end{aligned}$$

This implies that the assignment A' that we construct is better than assignment A. Thus, assignment A cannot be an optimal assignment.

Hence, the theorem is proven. ∎

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

Proof: As shown in Fig. 1, suppose in an assignment A_1, c_1 is assigned to s_1 and c_2 is assigned to s_2. We construct two assignments A_2 and A_3 based on A_1. A_2 assigns both c_1 and c_2 to s_1 and keeps the assigned servers of other clients unchanged. A_3 assigns both c_1 and c_2 to s_2 and keeps the assigned servers of other clients unchanged. In the following, we show that at least one assignment of A_2 and A_3 is better than A_1.



(a) Assignment A_1 (b) Assignment A_2 (c) Assignment A_3

Fig. 1. Diagram of Theorem 2

The total length of interaction paths in assignment A_1 is given by

$$\begin{aligned}
D(A_1) &= \sum_{c_i,c_j \in C} d_{A_1}(c_i, c_j) \\
&= \sum_{c_j \neq c_2} d_{A_1}(c_1, c_j) + \sum_{c_j \neq c_1} d_{A_1}(c_2, c_j) \\
&\quad + d_{A_1}(c_1, c_2) + \sum_{c_i,c_j \neq c_1,c_2} d_{A_1}(c_i, c_j). \qquad (1)
\end{aligned}$$

By analyzing the first term of (1), we have

$$\begin{aligned}
&\sum_{c_j \neq c_2} d_{A_1}(c_1, c_j) = d_{A_1}(c_1, c_1) + \sum_{c_j \neq c_1,c_2} d_{A_1}(c_1, c_j) \\
&= |C| \cdot d(c_1, s_1) + \sum_{c_j \neq c_1,c_2} \big(d(s_1, s_{A_1}(c_j)) + d(s_{A_1}(c_j), c_j)\big).
\end{aligned}$$

Similarly, for the second term of (1), we have

$$\begin{aligned}
&\sum_{c_j \neq c_1} d_{A_1}(c_2, c_j) \\
&= |C| \cdot d(c_2, s_2) + \sum_{c_j \neq c_1,c_2} \big(d(s_2, s_{A_1}(c_j)) + d(s_{A_1}(c_j), c_j)\big).
\end{aligned}$$

Also note that $d_{A_1}(c_1, c_2) = d(c_1, s_1) + d(s_1, s_2) + d(c_2, s_2)$. Thus, (1) becomes

$$\begin{aligned}
D(A_1) &= \sum_{c_j \neq c_1,c_2} \big(d(s_1, s_{A_1}(c_j)) + d(s_{A_1}(c_j), c_j)\big) \\
&\quad + \sum_{c_j \neq c_1,c_2} \big(d(s_2, s_{A_1}(c_j)) + d(s_{A_1}(c_j), c_j)\big) \\
&\quad + (|C| + 1) \cdot d(c_1, s_1) + (|C| + 1) \cdot d(c_2, s_2) + d(s_1, s_2) \\
&\quad + \sum_{c_i,c_j \neq c_1,c_2} d_{A_1}(c_i, c_j). \qquad (2)
\end{aligned}$$

Similarly, the total lengths of interaction paths in assignments A_2 and A_3 can be written as

$$\begin{aligned}
D(A_2) &= 2 \cdot \sum_{c_j \neq c_1,c_2} \big(d(s_1, s_{A_2}(c_j)) + d(s_{A_2}(c_j), c_j)\big) \\
&\quad + (|C| + 1) \cdot d(c_1, s_1) + (|C| + 1) \cdot d(c_2, s_1) \\
&\quad + \sum_{c_i,c_j \neq c_1,c_2} d_{A_2}(c_i, c_j),
\end{aligned}$$

and

$$\begin{aligned}
D(A_3) &= 2 \cdot \sum_{c_j \neq c_1,c_2} \big(d(s_2, s_{A_3}(c_j)) + d(s_{A_3}(c_j), c_j)\big) \\
&\quad + (|C| + 1) \cdot d(c_2, s_2) + (|C| + 1) \cdot d(c_1, s_2) \\
&\quad + \sum_{c_i,c_j \neq c_1,c_2} d_{A_3}(c_i, c_j).
\end{aligned}$$

Since all clients except c_1, c_2 are assigned to identical servers in A_1, A_2 and A_3, for any clients $c_i, c_j \neq c_1, c_2$, $d_{A_1}(c_i, c_j) = d_{A_2}(c_i, c_j) = d_{A_3}(c_i, c_j)$. Thus, we have

$$\begin{aligned} &D(A_2) + D(A_3) - 2D(A_1) \\ &= (|C| + 1)\big(d(c_1, s_2) + d(c_2, s_1) - d(c_1, s_1) - d(c_2, s_2)\big) \\ &\quad - 2d(s_1, s_2). \end{aligned}$$

Since u is on the shortest paths from c_1 to s_1 and from c_2 to s_2, we have $d(c_1, s_1) = d(c_1, u) + d(s_1, u)$ and $d(c_2, s_2) = d(c_2, u) + d(s_2, u)$. By the triangle inequality, it follows that

$$\begin{aligned} &D(A_2) + D(A_3) - 2D(A_1) \\ &\leq (|C| + 1)\big(d(c_1, u) + d(s_2, u) + d(c_2, u) + d(s_1, u) \\ &\quad - d(c_1, u) - d(s_1, u) - d(c_2, u) - d(s_2, u)\big) - 2d(s_1, s_2) \\ &< 0. \end{aligned}$$

This shows that the sum of $D(A_2)$ and $D(A_3)$ is smaller than two times of $D(A_1)$, implying at least one of $D(A_2)$ and $D(A_3)$ must be smaller than $D(A_1)$, and the corresponding assignment is better than A_1. Therefore, assignment A_1 cannot be an optimal assignment.

Hence, the theorem is proven. ■

Based on Theorem 2, we obtain the following corollary.

Corollary 1: If a client c_2 is located on the shortest path between another client c_1 and a server s_1, then any assignment assigning c_1 to s_1 and assigning c_2 to a different server cannot be an optimal assignment.

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

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

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

We show that the client assignment problem is NP-complete by a polynomial reduction from the partition problem which is known to be NP-complete [5]. The partition problem is defined as follows: Given a finite set of positive integers B, does there exist a subset $B' \subseteq B$ such that $\sum_{b \in B'} b = \sum_{b \in B - B'} b$?

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

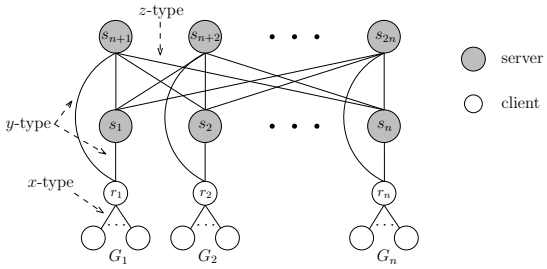


Fig. 2. Example of instance Q of the client assignment problem

In the network that we construct, servers s_i and s_{n+i} are on the shortest paths between the center client r_i and all other servers. According to Theorem 1, the assignments that assign the center client r_i to any server other than s_i or s_{n+i} cannot be optimal. Thus, in an optimal assignment, the center client of each group must be assigned to its connected server in either U_1 or U_2. In addition, each center client is on the shortest paths between all the other clients in the same group and all servers. According to Corollary 1, all clients in one group must be assigned to the same server in an optimal assignment. So, we only consider such assignments that assign all clients in group G_i to either server s_i in U_1 or server s_{n+i} in U_2.

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

For z-type links, their contributions depend on how many clients are assigned to server sets U_1 and U_2. The interaction path between each pair of clients contains one z-type link if the pair of clients are assigned to servers in different server

sets and contains two z-type links if the pair of clients are assigned to different servers in the same set. We denote by $C(s_i)$ the number of clients assigned to server s_i. Then, all z-type links contribute a total of

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

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

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

For each pair of servers $s_i \in U_1$ and $s_{n+i} \in U_2$, either $C(s_i) = |G_i| = b_i$ and $C(s_{n+i}) = 0$, or $C(s_i) = 0$ and $C(s_{n+i}) = |G_i| = b_i$. Therefore, for each possible assignment, $\sum_{s_i \in U_1} C(s_i)$ and $\sum_{s_i \in U_2} C(s_i)$ correspond to the sums of subsets B' and $B - B'$ respectively for a partitioning of B, and vise versa. It also follows that $\sum_{s_i \in U_1 \cup U_2} C(s_i) = \sum_{i=1}^{n} b_i = S$, $C(s_i)^2 + C(s_{n+i})^2 = b_i^2$ and $\sum_{s_i \in U_1} C(s_i) \cdot \sum_{s_i \in U_2} C(s_i) \leq \frac{S^2}{4}$. Thus, we have

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

The lower bound K is achieved only when $\sum_{s_i \in U_1} C(s_i) \cdot \sum_{s_i \in U_2} C(s_i) = \frac{S^2}{4}$, i.e., $\sum_{s_i \in U_1} C(s_i) = \sum_{s_i \in U_2} C(s_i) = \frac{S}{2}$. Therefore, there exists a client assignment such that the total length of interaction paths is at most K if and only if there exists a subset B' such that $\sum_{b \in B'} b = \sum_{b \in B - B'} b$ for the partitioning problem.

Hence, the theorem is proven. ∎

IV. Heuristic Algorithms

A brute-force solution to the client assignment problem is computationally expensive. There are a total of $|S|^{|C|}$ different assignment strategies for an exhaustive search, where $|S|$ is the number of servers and $|C|$ is the number of clients. The search space is huge even for small values of $|S|$ and $|C|$. We present three heuristic algorithms for the client assignment problem in this section, and then compare the performance of these algorithms via simulation experiments in Section V. These heuristic algorithms do not assume any particular routing strategy in the network. They compute the client assignment based simply on the network latencies between different nodes, which can be obtained with existing utilities like ping and King [6].

A. Nearest-Assignment

The first algorithm is an intuitive *Nearest-Assignment* algorithm, in which each client chooses its nearest server as its assigned server. The motivation of this approach is to minimize the latency between the client and its assigned server. *Nearest-Assignment* is easy to implement and can be performed in a distributed manner because each client selects its server independently. The computational complexity for each client to compute its nearest server is $O(|S|)$. When assuming that messages are routed along the shortest paths, we can show that the total interaction path length of *Nearest-Assignment* has an approximation ratio of 3 to an optimal assignment.

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

Proof: Consider two different clients u and v in the network. Let n_u and n_v be the nearest servers of clients u and v, respectively. Suppose in an optimal assignment A^*, client u is assigned to a server s_u and client v is assigned to a server s_v. We immediately have the following results

$$d(u, n_u) \leq d(u, s_u) \quad \text{and} \quad d(v, n_v) \leq d(v, s_v).$$

Therefore, by the triangle inequality, we have,

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

and

$$d(n_v, s_v) \leq d(v, n_v) + d(v, s_v) \leq 2d(v, s_v).$$

The length of the interaction path between u and v in *Nearest-Assignment* (denoted by A_N) is given by

$$d_{A_N}(u, v) = d(u, n_u) + d(n_u, n_v) + d(v, n_v).$$

By the triangle inequality, $d(n_u, n_v)$ should be shorter than the length of the concatenation of the paths from n_u to s_u, from s_u to s_v, and from s_v to n_v. Thus, it follows that

$$\begin{aligned} &d_{A_N}(u, v) \\ &\leq d(u, n_u) + d(n_u, s_u) + d(s_u, s_v) + d(n_v, s_v) + d(v, n_v) \\ &\leq d(u, s_u) + 2d(u, s_u) + d(s_u, s_v) + 2d(v, s_v) + d(v, s_v) \\ &= 3d(u, s_u) + d(s_u, s_v) + 3d(v, s_v) \\ &\leq 3d_{A^*}(u, v). \end{aligned}$$

For the interaction path from a client u to itself, it is easy to obtain

$$d_{A_N}(u, u) = 2d(u, n_u) \leq 2d(u, s_u) = d_{A^*}(u, u).$$

Therefore, for each pair of clients, the length of their interaction path in *Nearest-Assignment* is within three times of that in an optimal assignment. As a result, the total length of all interaction paths in *Nearest-Assignment* is within three times of that in an optimal assignment.

Hence, the theorem is proven. ■

B. Greedy-Assignment

The second algorithm is called *Greedy-Assignment*. *Greedy-Assignment* starts with an empty assignment A and continues to assign clients to servers until all clients have been assigned. Initially, the total length of all interaction paths is zero because no client is assigned to any server. At each step, *Greedy-Assignment* considers all possible pairs (c, s) of an unassigned client c and a server s to make a new assignment and computes the increase in the total interaction path length for each pair. The increase is given by the total length of interaction paths from this newly assigned client c to all the clients that have already been assigned, i.e.,

$$\sum_{c_i \in C' \cup \{c\}} d_A(c, c_i), \tag{3}$$

where C' is the set of clients already assigned. The pair of client and server that introduces the minimum total length of these interaction paths is selected to make the assignment. By analyzing (3), we have

$$\begin{aligned}
&\sum_{c_i \in C' \cup \{c\}} d_A(c, c_i) \\
&= 2d(c, s) + \sum_{c_i \in C'} \Big(d(c, s) + d\big(s, s_A(c_i)\big) + d\big(s_A(c_i), c_i\big) \Big) \\
&= (|C'| + 2) \cdot d(c, s) + \sum_{c_i \in C'} d\big(s, s_A(c_i)\big) \\
&\quad + \sum_{c_i \in C'} d\big(s_A(c_i), c_i\big).
\end{aligned} \tag{4}$$

If we denote by $m(s)$ the number of clients that have been assigned to server s, the second term of (4) can be written as $\sum_{c_i \in C'} d\big(s, s_A(c_i)\big) = \sum_{s_j \in S} m(s_j) \cdot d(s, s_j)$. Thus, it follows that:

$$\begin{aligned}
&\sum_{c_i \in C' \cup \{c\}} d_A(c, c_i) \\
&= (|C'| + 2) \cdot d(c, s) + \sum_{s_j \in S} m(s_j) \cdot d(s, s_j) \\
&\quad + \sum_{c_i \in C'} d\big(s_A(c_i), c_i\big).
\end{aligned} \tag{5}$$

Note that the third term of (5) does not change with the unassigned client c and server s selected to make the new assignment. Thus, for comparison purpose, at each step of *Greedy-Assignment* it can be omitted from the calculation of the increase in interaction path length. In addition, the second term of (5) is identical for all pairs of unassigned clients and a given server. Hence, for each server s, we first compute the second term of (5) for s and then compute the first term of (5) for s and each unassigned client c to find the pair of client and server producing the minimum increase in interaction path length. Since all possible client-server pairs are examined for each selection, *Greedy-Assignment* is a centralized algorithm that requires complete pair-wise latency information between all clients and servers. The pseudo code of *Greedy-Assignment* is presented in Fig. 3. At each step, the calculation of line 7 has a computational complexity of $O(|S|)$ for each server, and calculating the increase in the total path length from lines 8 to 13 for all clients has a complexity of $O(|C|)$. Therefore, the total time complexity of each step is $O(|S| \cdot (|S| + |C|))$. As a result, the overall complexity of *Greedy-Assignment* is $O\big((|S| + |C|) \cdot |S| \cdot |C|\big)$.

```
1:  C' ← ∅;
2:  for all s ∈ S do
3:      m(s) ← 0;
4:  while C' ≠ C do
5:      min ← ∞;
6:      for all s ∈ S do
7:          t2 ← Σ_{sj∈S} m(sj) · d(s, sj);
8:          for all c ∈ C − C' do
9:              len ← (|C'| + 2) · d(c, s) + t2;
10:             if len < min then
11:                 min ← len;
12:                 c* ← c;
13:                 s* ← s;
14:     set sA(c*) = s*;
15:     C' ← C' ∪ {c*};
16:     m(s*) ← m(s*) + 1;
```

Fig. 3. The *Greedy-Assignment* algorithm

C. Distributed-Greedy-Assignment

Distributed-Greedy-Assignment is the third algorithm we propose. It also adopts a greedy approach. However, different from *Greedy-Assignment* which does not allow clients to change their servers once assigned, *Distributed-Greedy-Assignment* starts with an existing assignment and continues to modify the assignment for reducing the total length of interaction paths. To perform modification, each client examines all servers to determine whether assigning itself to a different server would decrease the total length of interaction paths. If so, the client changes its assigned server to the server that decreases the total path length most. Otherwise, the client keeps the currently assigned server. This process continues until no client can reduce the total length of interaction paths by changing its assigned server. Since the assignment modification can only decrease the total path length, the resultant assignment cannot be worse than the initial assignment.

In deciding whether to change or keep its assigned server, a client c only needs to consider the interaction paths involving itself since the interaction paths between all other clients do not change with the assignment modification of this client. Specifically, for each server s, client c calculates the decrease

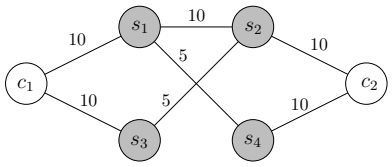


Fig. 4. An example in which changing two clients' assigned servers simultaneously increases the total interaction path length

in the total length of interaction paths between itself and all clients, supposing that it is assigned to s. Similar to (5), this total length is given by $\sum_{c_i \in C} d_A(c, c_i) = (|C|+1)d(c,s) + \sum_{s_j \in S} m(s_j)d(s,s_j) + \sum_{c_i \in C} d(s_A(c_i), c_i)$, where the third item is independent of c's assignment modification and can be omitted from the calculation for comparison purpose. Therefore, the calculation only needs the information of the latency $d(c,s)$ between c and s, the latencies between s and other servers, and the number of clients $m(s_j)$ that has been assigned each server s_j. To this end, each server maintains the latencies from itself to other servers, and keeps track of how many clients have been assigned to itself. The client considering assignment modification contacts each server to measure the latency to the server and collects the inter-server latencies and the number of assigned clients that are maintained by the server. After all servers have been contacted, the client then performs the above calculation and assignment modification locally. In our experiments, we choose *Nearest-Assignment* as the initial assignment for *Distributed-Greedy-Assignment* to begin with. Thus, the whole *Distributed-Greedy-Assignment* algorithm is performed in a distributed manner.

One issue with *Distributed-Greedy-Assignment* is that, if two or more clients change their assigned servers simultaneously, the total interaction path length is not guaranteed to decrease because the calculation of each client is based on the assumption that the assigned servers of other clients remain unchanged. Fig. 4 gives an example, where the network contains clients c_1, c_2 and servers s_1, s_2, s_3, s_4, and the number on each link represents the length of the link. The initial assignment assigns client c_1 to server s_1 and client c_2 to server s_2. The lengths of interaction paths between client pairs (c_1, c_2), (c_1, c_1) and (c_2, c_2) are 30, 20 and 20 respectively. So, the total path length is 70. When client c_1 tries to change its assigned server, it would choose s_3 since the length of the interaction path between c_1 and c_2 would be reduced to 25 while the lengths of interaction paths between client pairs (c_1, c_1) and (c_2, c_2) stay the same, assuming that c_2 keeps its assigned server unchanged. Similarly, client c_2 would choose to change its assigned server to s_4. However, if the two clients both change their assigned servers at the same time, the path length between c_1 and c_2 would be increased to 40 which is even longer than the initial assignment. Therefore, a concurrency control mechanism is required to prevent clients from performing assignment modifications at the same time. To guarantee that assignment modification proceeds one client at a time, a token can be circulated among clients and only the client obtaining the token is allowed to perform assignment modification. We refer to a complete circulation in which each client receives the token once and performs one assignment modification as an *iteration*. Since the computational complexity of an assignment modification for one client is $O(|S|^2)$, the total computational complexity of one iteration for all clients is $O(|C||S|^2)$. Our experimental results in Section V show that over 90% of the potential reduction in interaction path length is normally achieved in the first iteration of assignment modification. Thus, a practical strategy is to execute the *Distributed-Greedy-Assignment* for just one iteration.

D. Dealing with Limited Server Capacities

So far, we have not assumed any limitation on the capacity of each server. In practice, servers have finite capacities. If the number of clients assigned to a server exceeds its capacity, the processing delay at the server can increase significantly [8]. Now, we discuss how to take simple steps in our proposed assignment algorithms to ensure that the capacity of each server is not exceeded. In *Nearest-Assignment*, when a client searches for its nearest server, if this server is saturated, the client turns to the second nearest server and keeps doing so until it finds a server that can accommodate more clients. In *Greedy-Assignment*, the algorithm only considers the servers that are not saturated to make the new assignment at each step. For *Distributed-Greedy-Assignment*, each client would only consider to change to servers that are not saturated in the assignment modification process. We evaluate these modified algorithms in Section V.

V. Experimental Evaluation

To evaluate the proposed assignment algorithms, we have conducted extensive simulation experiments parameterized with several sets of real and synthetic network latency data including Meridian [15], Waxman [12] and Random [2]. The experimental results showed similar performance trends for different data sets. Due to space limitations, we report only the representative results for the Meridian data set of real Internet latency in this section. The Meridian data set contains pairwise latency measurements between 2500 nodes in the Internet using the King measurement technique [6]. The measurements for some node pairs are invalid and the nodes involved in these measurements were excluded from the experiments. As a result, the network simulated in our experiments is represented by a latency matrix between 1796 nodes. A client is assumed to be located at each node and a certain number of servers are randomly placed in the network. In the default parameter setting, the number of servers is set at 80.

To quantify the relative performance difference, the experimental results of different algorithms are normalized with respect to a theoretical lower bound on the total length of interaction paths, which is derived as follows. Note that for any client assignment A, the interaction path between two clients

u, v has a length of

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

By adding up the above inequalities for all client pairs, we obtain the following lower bound on the total length of all interaction paths.

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

Note that this lower bound is a super bound that may not be achievable by any client assignment. This is because the above lower bound calculation does not enforce the constraint that each client is assigned to only one server through which it interacts with all the other clients. The total length of interaction paths produced by each algorithm normalized by the above bound shall be called the *normalized interactivity*. For each parameter setting, we performed 1000 simulation runs using 1000 different sets of servers selected at random. The average performance of these simulation runs is plotted for performance comparison.

A. Impact of Number of Servers

Fig. 5a shows the average normalized interactivity produced by the three heuristics for different numbers of servers. In this experiment, we do not assume any limitation on server capacities. As can be seen, the two greedy algorithms outperform the *Nearest-Assignment* algorithm, and they generally perform close to the lower bound. For all numbers of servers tested, the total interaction path lengths of *Greedy-Assignment* and *Distributed-Greedy-Assignment* are on average within 47% and 35% of the lower bound respectively, which means that they are within at most the same percentage of the results of the optimal assignment. Comparing the two greedy algorithms, *Distributed-Greedy-Assignment* consistently outperforms *Greedy-Assignment* over different server numbers.

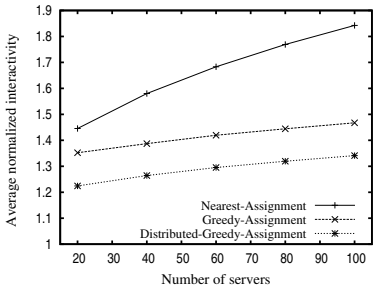
To further study the improvements of the two greedy algorithms over the *Nearest-Assignment* algorithm, we plot the average and the 90th percentile improvements of the 1000 simulation runs in Fig. 5b. For each simulation run, we compute the improvements of the greedy algorithms by subtracting their total path lengths from the total path length of *Nearest-Assignment* and normalizing the difference by the total path length of *Nearest-Assignment*. As shown in Fig. 5b, the improvements of the greedy algorithms over *Nearest-Assignment* increase as the number of servers grows. The average improvements of *Greedy-Assignment* and *Distributed-Greedy-Assignment* are about 12% and 20% when the number of servers is 40, and they increase to 20% and 27% for 100 servers. In general, when there are more servers in the network, the latency between the client and its nearest server reduces. Thus, the increasing trend in the improvements of the greedy algorithms over *Nearest-Assignment* implies that reducing the client-server latency alone is not the most efficient way of improving the interactivity. The 90th percentile results indicate that in $1/10$ of the simulation runs executed, *Greedy-Assignment* and *Distributed-Greedy-Assignment* outperform *Nearest-Assignment* by at least 21% and 26% for 40 servers, and 27% and 32% for 100 servers. Note that *Distributed-Greedy-Assignment* uses *Nearest-Assignment* as its initial assignment. Thus, the client assignments generated by *Distributed-Greedy-Assignment* cannot be worse than *Nearest-Assignment*. However, unlike *Distributed-Greedy-Assignment*, *Greedy-Assignment* is not guaranteed to outperform *Nearest-Assignment* for all cases. In the experiments, we observe that when the number of servers is 20, *Greedy-Assignment* generates worse results than *Nearest-Assignment* in about $1/10$ of the simulation runs. *Greedy-Assignment* becomes better than *Nearest-Assignment* for all simulation runs when there are 100 servers in the network.

In our experiments, *Distributed-Greedy-Assignment* is terminated when no client can further reduce the total interaction path length by changing its assigned server. To investigate the convergence speed of *Distributed-Greedy-Assignment*, we plot in Fig. 5c the average improvements of *Distributed-Greedy-Assignment* over *Nearest-Assignment* after each iteration when there are 80 servers in the network. The results show that, *Distributed-Greedy-Assignment* converges quickly in just few iterations. In particular, about 90% of the improvements are made in the first iteration, and over 99% of the improvements are achieved after three iterations. Similar trends have also been observed in the experimental results with other server numbers. Thus, a practical strategy is to execute the *Distributed-Greedy-Assignment* algorithm for one iteration only.

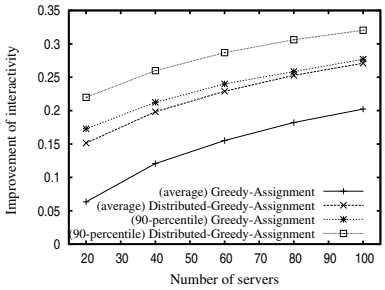
B. Impact of Server Capacity

In this experiment, we study the effect of limited server capacity on the performance of the proposed algorithms. Fig. 6a shows the average normalized interactivity as a function of the maximum number of clients that can be assigned to each server. Note that the theoretical lower bound does not change with server capacity as it assumes unlimited server capacity. It can be seen from Fig. 6a that the interaction path lengths generated by all algorithms increase with decreasing server capacity. This is because at low server capacities, the algorithms may not be able to assign the clients to the best servers based on network latency due to insufficient server capacities. The relative performance of the three algorithms remain similar over different server capacities. The performance of the greedy algorithms is better than that of *Nearest-Assignment* for all server capacities tested.

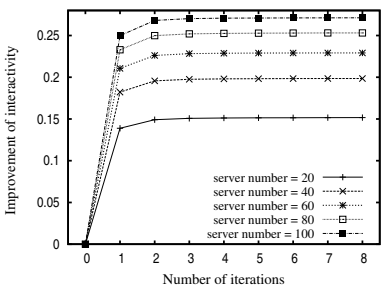
Fig. 6b shows the average and 90th percentile improvements of the greedy algorithms over *Nearest-Assignment*. As can be seen, the improvements decrease when each server can accommodate fewer clients. On average, *Greedy-Assignment* and *Distributed-Greedy-Assignment*'s improvements are 15% and 22% respectively when the server capacity is 250, while the improvements drop to 10% and 16% when the server capacity decreases to 100. These results imply that limited server capacity has greater effect on the performance of



(a) Average normalized total length of interaction paths

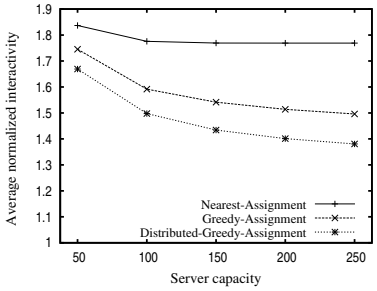


(b) Improvement in the total length of interaction paths

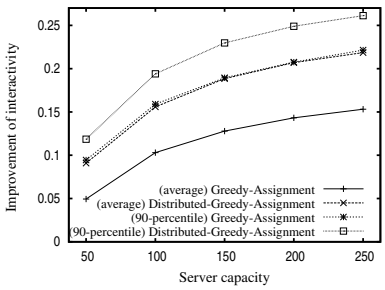


(c) Impact of iteration number on the performance of *Distributed-Greedy-Assignment*

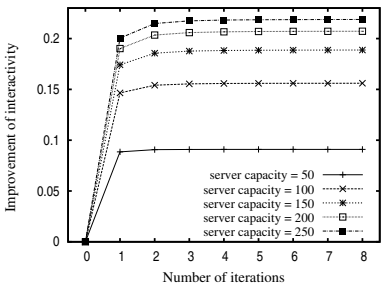
Fig. 5. Experimental results for different numbers of servers



(a) Average normalized total length of interaction paths



(b) Improvement in the total length of interaction paths



(c) Impact of iteration number on the performance of *Distributed-Greedy-Assignment*

Fig. 6. Experimental results for different server capacities

the greedy algorithms than *Nearest-Assignment*. The results also suggest that increasing the capacities of the servers at strategic locations has the potential to significantly improve the interactivity.

Fig. 6c shows the average improvement of *Distributed-Greedy-Assignment* over *Nearest-Assignment* as a function of the number of iterations executed. For all server capacities tested, *Distributed-Greedy-Assignment* achieves about 90% of its improvements in the first iteration, similar to the observations made in Fig. 5c. Therefore, it is generally sufficient to execute *Distributed-Greedy-Assignment* for just one iteration.

VI. Conclusion

In this paper, we have investigated the client assignment problem for enhancing the interactivity of DIAs. We have formulated the problem as a combinational optimization problem on graphs and shown that this problem is NP-complete. Three heuristic algorithms have been proposed and experimentally evaluated using real Internet latency data. The results show that the proposed greedy algorithms perform close to the optimal assignment, and generally outperform the *Nearest-Assignment* algorithm that assigns each client to its nearest server.

References

[1] 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.

[2] K. Calvert and E. Zegura. GT internetwork topology models, 1997.

[3] E. Cronin et al. Constrained mirror placement on the internet. *IEEE J. Sel. Areas Commun.*, 20(7):1369–1382, 2002.

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

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] BM Waxman. Routing of multipoint connections. *IEEE J. Sel. Areas Commun.*, 6(9):1617–1622, 1988.

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

[14] 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.

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