2012 IEEE 18th International Conference on Parallel and Distributed Systems

An Enhanced Genetic Algorithm for Server Placement in Distributed Interactive Applications

Hanying Zheng and Xueyan Tang
School of Computer Engineering
Nanyang Technological University
Singapore 639798
{*zhen0084, asxytang*}*@ntu.edu.sg*

***Abstract*—Recent years have witnessed the enormous popularity of distributed interactive applications (DIAs), which allow participants that are distributed in the network to interact with each other concurrently. The rapid growth of DIAs has raised stringent requirements on providing realistic sense of interaction between participants, whose quality is heavily influenced by network latencies. Although network latencies cannot be eliminated due to geographical spreads of participants, it is possible to reduce them by a smart selection of the locations where the servers of the DIAs are placed. The locations of servers affect not only the inter-server latencies but also the latencies from participants to servers, both of which are involved in the interactions among participants. Thus, the placement of servers is an important factor to the interactivity performance of DIAs. We formulate the server placement problem, and propose to solve it by an enhanced genetic algorithm, whose genetic operators are specially designed based on the nature of the problem. Experimental results using various datasets show that our algorithm leads to appreciable improvement of the interaction quality in DIAs.**

***Keywords*-server placement; distributed interactive application; network latency; optimization; genetic algorithm**

I. Introduction

There has been a tremendous growth in the popularity of Distributed Interactive Applications (DIAs) in recent years. DIAs are human-in-the-loop networked environments where participants scattered in different places can interact with each other concurrently to pursue some given tasks such as networked gaming, collaborative document editing and collaborative e-learning [15]. In networked gaming, for example, avatars, controlled by human players, assemble in an online virtual world and interact with one another in real time. DIAs are usually implemented with client-server architectures. Clients, which refer to the computing devices that render the application state to the participants, exchange with the servers messages encapsulating the operations issued by the participants or the updates of the application state so that consistent states of the DIA are maintained at all clients.

The networked-servers model is the most common client-server architecture for DIAs because it provides good scalability, robustness as well as better performance than the single-server model. In the networked-servers model, there are a group of interconnected servers, each of which maintains a copy of the state of the DIA. Each client is assigned to exactly one of the servers, and interacts with other clients through its assigned server. When issuing an operation, a client sends the operation to its assigned server, which then relays it to the rest servers. On receiving the operation, the servers update their states accordingly and deliver the resultant state update to the clients assigned to them. The duration from the time when a client u sends out an operation to the time when another client v receives the state update due to u's operation from its assigned server is known as the interaction time from u to v.

DIAs are latency sensitive. The quality of interaction between clients is heavily influenced by network latencies. It has been shown in [2] that latencies around 100ms are already noticeable in networked games and it becomes annoying when latencies increase to above 200ms. Fast-paced games have even higher requirements on network latency. Therefore, guaranteeing the quality of interaction between clients is a critical challenge for DIAs.

Approaches such as local-lag [4] and dead reckoning [10] are animatedly discussed over last decade to cope with the impact of network latencies. However, these approaches could not actually reduce latencies. In contrast, they alleviate the impact of latencies on consistency by introducing additional delays or computational overheads. For example, the local-lag approach prolongs the waiting time before an operation is executed at each server so as to maintain the same order of operation execution at different servers.

Different from the above approaches, we note that network latencies heavily depend on the locations where the servers of the DIA are placed in the network. Any change of a server location alters the physical connections incident to the server and therefore influences the network latencies involved in the interactions. Thus, the placement of servers is an important factor to the quality of interaction between clients. In this paper, we explore the problem of how to place multiple servers in the network, with the goal of minimizing the interaction times among all clients.

The rest of this paper is organized as follows. Section II gives a brief overview of related works. Section III formulates the server placement problem. Section IV pro-

1521-9097/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPADS.2012.86

poses an enhanced genetic algorithm for server placement. Experimental results of the algorithm in Section V show that the algorithm leads to appreciable improvement of the interaction quality between clients. Finally, Section VI concludes the paper.

II. Related Work

To our knowledge, there are limited works discussing the placement of servers in the field of DIA.

Zoom-In-Zoom-Out (ZIZO) is a distributed algorithm presented by Lee et al. [5] for server selection in large scale networked games. Given a set of pre-deployed servers and a set of clients in interaction, the purpose of ZIZO is to find a minimum subset of servers for client interactions such that the maximum interaction time between any client pairs satisfies a given real-time delay requirement. Since it does not attempt to minimize the interaction time, the ZIZO algorithm can only provide bounded interaction quality.

Laoutaris et al. [12] propose a distributed algorithm to place service facilities to address the k-median problem that aims at minimizing the distances between clients and their closest facilities in large scale networks. In their algorithm, each node does not require global knowledge of the network, but demands only the information about the neighbourhood within certain hops from it. By reducing the amount of information required for each node, such algorithm overcomes the scalability limitation compared to the centralized solution.

Briceno et al. [9] solve the server selection problem that aims to promote some clients as secondary servers to offload the computation from the pre-deployed central server while minimizing the maximum response time of operations. This model however introduces additional message transmission delays between the central server and secondary servers to maintain the consistency of game states. Therefore, the overall interaction time among clients may not be optimised.

If the servers are already placed in the network, the problem of deciding how to assign clients to servers is referred to as the client assignment problem. Zhang et al. [17], [18] formally define the client assignment problems with the objective of minimizing the interaction times between clients as combinatorial optimization problems. They show the NP-completeness of the problems, and present centralized and distributed greedy solutions that generate assignments close to the optimal assignment.

A variant of the client assignment problem is the partitioning problem in distributed virtual environments [6] whose objective is to dynamically partition the game world into disjoint regions and assign these regions to different servers so as to balance the computation workload and reduce the inter-server communication cost. Lui el al. [6] develop a centralized version heuristic and a parallel version heuristic to address this problem. Ta et al. [13], [14] investigated the server placement problem under the partitioning approach. It is assumed that only the clients connecting to the same server are allowed to interact with each other. As a result, only the client-server latency is considered as the objective of optimization. This approach restricts the extent of interaction among clients.

III. Problem Formulation

The underlying network of the DIA is modelled as a graph $G = (V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is the set of links between nodes. The weight of a link represents the message transmission delay through this link. The path from a node x to another node y is the concatenation of links involved in the message transmission from x to y. The length of this path, denoted by $d(x, y)$, is the total weight of all the intermediate links along the path, representing the total transmission delay from x to y. It is worth noting that $d(x, x) = 0$ for any node x trivially.

We assume that the networked-servers model is used. If the servers are given in the network, each client should be assigned to exactly one of the servers for interacting with other clients. Let $C \subseteq V$ be the set of clients of the DIA. For each client $u \in C$, we denote the assigned server of u by s_u. The interaction between a client u that sends out an operation and another client v that receives the resultant state update due to u's operation involves three paths: the path from u to u's assigned server s_u, the path from s_u to v's assigned server s_v, and the path from s_v to v. We denote the concatenation of these three paths as the *interaction path* from client u to client v. Then, the *interaction path length* from u to v is calculated by $d(u, s_u) + d(s_u, s_v) + d(s_v, v)$, which represents the interaction time from u to v.

Suppose that each time a client generates an operation, the resultant state update is distributed to all the clients in the network to enable full interactions among all pairs of clients. Then, the overall quality of interaction in the DIA can be measured by the arithmetic mean M of the interaction path lengths between all client pairs, i.e.,

$$M = \frac{1}{|C|^2} \sum_{u \in C} \sum_{v \in C} \left(d\left(u, s_u\right) + d\left(s_u, s_v\right) + d\left(s_v, v\right) \right).$$

M reflects the average duration required for any client to keep track of any change of the application state. The lower the value of M is, the better the interactivity performance of the DIA is. Since $\frac{1}{|C|^2}$ is constant given the set of clients, minimizing the average interaction path length M is equivalent to minimizing the total interaction path length between all client pairs. Therefore, the server placement problem is formulated as follows.

Definition 1. *Given a network $G = (V, E)$, and a set of clients $C \subseteq V$. Let $S \subseteq V$ be the set of candidate server locations, each of which could be selected to place a server. The objective of the server placement problem is to find a set of k locations $S' \subseteq S$ to place servers for the DIA such that they are applicable to achieving the minimum value of the*

total interaction path length between all client pairs. That is, by placing the k servers at S' in the network, there exists a server $s_c \in S'$ for each client c such that

$$\sum_{c \in C} \sum_{c' \in C} \left(d\left(c, s_c\right) + d\left(s_c, s_{c'}\right) + d\left(s_{c'}, c'\right)\right)$$

is the minimum value of all possible total interaction path lengths.

Figure 1 shows an example of the server placement problem. Three nodes v_1, v_2 and v_3 in the network are interconnected via the links from v_1 to v_2 and from v_2 to v_3 whose weights are 10 and 1 respectively. There is a client located at each node, and all the three nodes can be selected to place servers. Suppose we would like to place two servers in this network. In this instance of the server placement problem, if v_1 and v_2 are selected as the solution, then the total interaction path length could be made as low as 46. This could be achieved by an assignment that assigns the clients at v_1, v_2 and v_3 to the servers at v_1, v_2 and v_2 respectively (see Figure 1(a)). If v_1 and v_3 are selected as the solution, then the best total interaction path length is 50, which is achieved when we assign the clients at v_1, v_2 and v_3 to the servers at v_1, v_3 and v_3 respectively (see Figure 1(b)). Similarly, if the servers are placed at v_2 and v_3, then the best total interaction path length is 64. It is achievable by assigning the clients at v_1, v_2 and v_3 to the servers at v_2, v_2 and v_3 respectively (see Figure 1(c)). Therefore, among these three possible server placements, placing servers at nodes v_1 and v_2 are applicable to achieving the minimum value of the total interaction path length. Hence, the best server placement solution is $S' = \{v_1, v_2\}$.

We remark that our server placement problem is significantly different from the classical graph-theoretic k-median problem which is commonly used to model server placement for web content distribution in the Internet [8]. The k-median problem aims to place k servers in the network to minimize the total distance from all the clients to their closest servers. Since the k-median problem does not consider the inter-server latencies, its solution cannot ensure the optimization of the interaction quality between clients. An example is depicted in Figure 2. In this network, there are four clients located at nodes $\{c_1, c_2, c_3, c_4\}$, and four candidate server locations $\{v_1, v_2, v_3, v_4\}$. Suppose that the number of servers to place is 2. Then, the optimal solution to the k-median problem is $\{v_1, v_2\}$. In this case, the best total interaction path length is 96 and is achieved when assigning all clients to their respective nearest servers as shown in Figure 2(a). However, the optimal solution to our server placement problem is to place servers at locations v_3 and v_4 instead. In this case, the total interaction path length can be minimized to 72, which is achievable by assigning all clients to their respective nearest servers (see Figure 2(b)). This results in a 25% improvement in the quality of interaction between clients compared to the solution of the k-median problem.

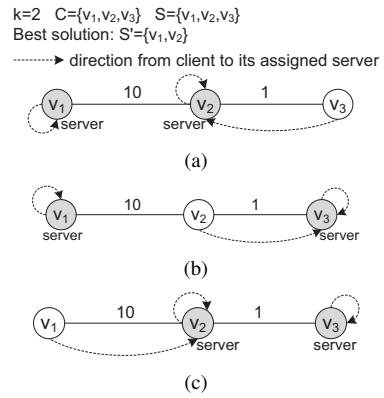
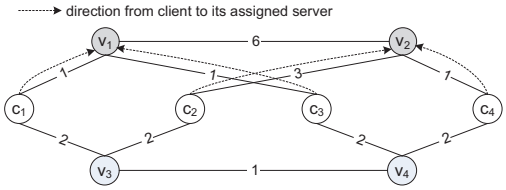
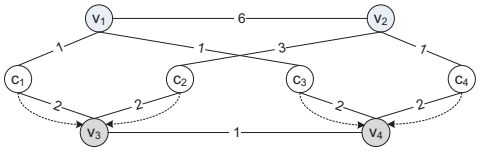


Figure 1. An example of the server placement problem



(a) Client assignment in the optimal solution of the k-median problem



(b) Client assignment in the optimal solution of the server placement problem

Figure 2. An example showing the difference between the k-median problem and our server placement problem

IV. Enhanced Genetic Algorithm

For larger than toy-size server placement problems, exhaustive search is infeasible because the search space could be extremely huge. Given a set of candidate server locations S in the network, if k server locations are to be selected out of all $|S|$ candidates, the number of possible server placement strategies is $\binom{|S|}{k}$. Meanwhile, with a set of clients C, since there are $k^{|C|}$ possible client assignment strategies given a server placement, it is only after a complete searching of $\binom{|S|}{k} \cdot k^{|C|}$ different solutions that a global optimum can be found. Intuitive heuristics such as local search and greedy algorithms may offer decent solutions but are unreliable if getting stuck in a local optimum. In this section, we propose an enhanced genetic algorithm, whose genetic operators are specially designed based on the nature of the server placement problem.

A. Chromosome Representation

We use binary encoding to present the placement of servers in a chromosome. Each chromosome o is represented as a string of bits $(g_1^o, g_2^o, ..., g_{|S|}^o)$, where $|S|$ is the number of candidate server locations, and the j-th bit g_j^o refers to the j-th candidate server location. Each bit takes the value of 1 if its corresponding candidate location is selected to place a server, and takes the value of 0 otherwise. There are exactly k bits with value 1 in each chromosome, where k is the number of candidate locations to be selected.

B. Initialization

In the initialization stage of our enhanced genetic algorithm, a population of p chromosomes is generated in the following way. For each chromosome o_i $(1 \leq i \leq p)$, we randomly pick a subset of $\frac{1}{2}|C|$ clients from the entire client set C according to a uniform distribution and then we run a greedy k-median subroutine based on this subset to select k locations out of the candidate server location set. The greedy k-median subroutine is originally designed for the classical k-median problem. It starts from an empty set of server locations. In each iteration, the subroutine selects a new candidate server location that yields the largest reduction in the total distance from all clients to their nearest servers. It stops when k server locations are selected.

```
1:  S' ← ∅;
2:  for all c ∈ C' do
3:      s_c ← ∅;
4:  for i = 1 to k do
5:      D* ← ∞; //minimum total client-server distance
6:      for all s ∈ S \ S' do
7:          D ← 0; //total client-server distance for testing s
8:          for all c ∈ C' do
9:              if s_c = ∅ or d(c, s_c) > d(c, s) then
10:                 D ← D + d(c, s);
11:             else
12:                 D ← D + d(c, s_c);
13:         if D* > D then
14:             D* ← D, s* ← s;
15:     S' ← S' ∪ {s*};
16:     for all c ∈ C' do
17:         if s_c = ∅ or d(c, s_c) > d(c, s*) then
18:             s_c ← s*;
19: output S';
```

Algorithm 1: Greedy k-median Subroutine

Algorithm 1 shows the pseudo code of the subroutine. The input is a client subset C' and the output is a set S' containing k server locations. To avoid repeatedly comparing the distances between each client c and all the servers already placed when testing a new candidate server location (lines 9 to 12), we maintain the nearest server of c in a variable s_c after each iteration (lines 16 to 18). This reduces the time complexity of testing a new candidate server location to $O(|C'|)$. Since $|C'| = \frac{1}{2}|C|$, the overall time complexity of the greedy k-median subroutine is $O(k|C||S|)$.

Here, we prove that when applying the greedy k-median subroutine to the server placement problem for DIAs, its result can be bounded within an approximation ratio of $(1+2\alpha)$ for networks satisfying the triangle inequality property, where α is the approximation ratio of the greedy k-median subroutine to the original k-median problem.

We first show the following theorem.

Theorem 1. *For networks satisfying the triangle inequality, the optimal placement strategy of the k-median problem produces a solution with an approximation ratio of 3 for the server placement problem for DIAs.*

Proof: In a network satisfying the triangle inequality, let $C = \{c_1, c_2, ..., c_n\}$ be a set of clients, and S be a set of candidate server locations in the network. For the server placement problem, we denote by $S_P \subseteq S$ and $p_i \in S_P$ respectively the set of k server locations and the assigned server of each client c_i $(1 \leq i \leq n)$ that minimize the total interaction path length between all client pairs. For the k-median problem, we denote by $S_M \subseteq S$ and $m_i \in S_M$ respectively the set of k server locations and the nearest server of each client c_i $(1 \leq i \leq n)$ that minimize the total distance from all the clients to their nearest servers.

If we apply the optimal strategy of the k-median problem to the server placement problem, we obtain the following total interaction path length

$$\begin{aligned} T_M &= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(d\left(c_i, m_i\right) + d\left(m_i, m_j\right) + d\left(m_j, c_j\right)\right) \\ &= 2n\sum_{i=1}^{n} d\left(c_i, m_i\right) + \sum_{i=1}^{n}\sum_{j=1}^{n} d\left(m_i, m_j\right). \end{aligned}$$

By the triangle inequality, we have

$$d(m_i, m_j) \leq d(c_i, m_i) + d(c_i, c_j) + d(c_j, m_j).$$

Therefore,

$$\begin{aligned} T_M &\leq 2n\sum_{i=1}^{n} d\left(c_i, m_i\right) \\ &\quad + \sum_{i=1}^{n}\sum_{j=1}^{n}\left(d\left(c_i, m_i\right) + d\left(c_i, c_j\right) + d\left(c_j, m_j\right)\right) \\ &= 4n\sum_{i=1}^{n} d\left(c_i, m_i\right) + \sum_{i=1}^{n}\sum_{j=1}^{n} d\left(c_i, c_j\right). \end{aligned}$$

Since $\sum_{i=1}^{n} d(c_i, m_i)$ is the minimum total distance from all the clients to their nearest servers among all server placement strategies, we have

$$\begin{aligned} 4n\sum_{i=1}^{n} d(c_i, m_i) &\leq 4n\sum_{i=1}^{n} d(c_i, p_i) \\ &\leq 2(2n\sum_{i=1}^{n} d(c_i, p_i) + \sum_{i=1}^{n}\sum_{j=1}^{n} d(p_i, p_j)) \end{aligned}$$

$$= 2T_P, \quad (1)$$

where T_P is the minimum total interaction path length achievable in the server placement problem. According to the triangle inequality, we also have

$$d(c_i, c_j) \leq d(c_i, p_i) + d(p_i, p_j) + d(p_j, c_j).$$

Thus,

$$\begin{aligned} T_M &\leq 2T_P + \sum_{i=1}^{n}\sum_{j=1}^{n} d\left(c_i, c_j\right) \\ &\leq 2T_P + \sum_{i=1}^{n}\sum_{j=1}^{n} \left(d\left(c_i, p_i\right) + d\left(p_i, p_j\right) + d\left(p_j, c_j\right)\right) \\ &= 2T_P + T_P = 3T_P. \end{aligned}$$

Hence, the theorem is proven. ■

Based on Theorem 1, we obtain the following corollary.

Corollary 1. *If the greedy k-median subroutine has an approximation ratio of α for the k-median problem, then it has an approximation ratio of $(1 + 2\alpha)$ for the server placement problem for DIAs.*

Proof: The proof is similar to that of Theorem 1, except that a factor α needs to be added to inequality (1). ■

Although deriving the exact value of α is out of the scope of this paper, Corollary 1 implies that the combination of k server locations output from the greedy k-median subroutine is applicable to achieving a bounded total interaction path length among the input $\frac{1}{2}|C|$ clients, and is therefore expected to be closer to the optimum in the search space than randomly selected server combinations. By adopting such server placement to initialize the chromosomes in the population, each individual could be preeminent and hence the convergence time of the genetic algorithm could be significantly reduced.

C. Fitness Evaluation

For each chromosome, given its server locations, a measure of the best achievable total interaction path length is required in order to evaluate its fitness. To this end, we need to solve the client assignment problem given the server locations which has been proven to be NP-complete in [17].

We present here a greedy assignment subroutine to assign clients to servers. The pseudo code is shown in Algorithm 2. The input of the subroutine includes a set of server locations $S' \subseteq S$ ($|S'| = k$) to be evaluated and the entire client set C. The output includes a client assignment strategy and the corresponding total interaction path length T. The subroutine starts with an empty assignment. In each iteration (lines 6 to 18), it assigns an unassigned client to a server such that this new assignment results in the minimum increase in the total interaction path length among all possible assignments of the unassigned clients.

```
 1: T ← 0;
 2: for all s ∈ S' do
 3:     T_1^(2)(s) ← 0, p(s) ← 1;
 4: for all c ∈ C do
 5:     assigned[c] ← false;
 6: for i = 1 to |C| do
 7:     T_min ← ∞;
 8:     for all s ∈ S' do
 9:         while assigned[L[s][p(s)]] is true do
10:             p(s) ← p(s) + 1;
11:         T_i^(1)(c, s) ← 2i · d(L[s][p(s)], s);
12:         if T_i^(1)(c, s) + T_i^(2)(s) < T_min then
13:             c_i^* ← L[s][p(s)], s_{c_i^*} ← s;
14:             T_min ← T_i^(1)(c, s) + T_i^(2)(s);
15:     assigned[c_i^*] ← true;
16:     T ← T + T_min + 2(|C| − i) · d(c_i^*, s_{c_i^*});
17:     for all s ∈ S' do
18:         T_{i+1}^(2)(s) = T_i^(2)(s) + 2d(s, s_{c_i^*});
19: return T;
```

Algorithm 2: Greedy Assignment Subroutine

Suppose that at the beginning of the i-th iteration, there is a set of clients $\{c_1^*, c_2^*, ..., c_{i-1}^*\} \subset C$ already assigned to the servers where c_j^* is the client assigned in the j-th iteration. Let $s_{c_j^*}$ denote the assigned server of client c_j^*. For each unassigned client c and each server $s \in S'$, the increase in the total interaction path length if c is assigned to s in the i-th iteration can be calculated by

$$\begin{aligned} T_i(c, s) &= 2 \cdot d\left(c, s\right) \\ &\quad + 2 \cdot \sum_{j=1}^{i-1} (d(c, s) + d(s, s_{c_j^*}) + d(s_{c_j^*}, c_j^*)) \\ &= 2i \cdot d(c, s) + 2 \cdot \sum_{j=1}^{i-1} d(s, s_{c_j^*}) + 2 \cdot \sum_{j=1}^{i-1} d(s_{c_j^*}, c_j^*). \end{aligned}$$

Let

$$T_i^{(1)}(c, s) = 2i \cdot d\left(c, s\right),$$

$$T_i^{(2)}(s) = 2 \cdot \sum_{j=1}^{i-1} d(s, s_{c_j^*}),$$

and

$$T_i^{(3)} = 2 \cdot \sum_{j=1}^{i-1} d(s_{c_j^*}, c_j^*).$$

Then,

$$T_i(c, s) = T_i^{(1)}(c, s) + T_i^{(2)}(s) + T_i^{(3)}. \quad (2)$$

Note that the third term $T_i^{(3)}$ does not involve either c or s. Thus, for comparison purpose, we can omit this term in the calculation of $T_i(c, s)$ because all possible new assignments in this iteration generate the same value of it.

The second term $T_i^{(2)}(s)$ does not involve the unassigned client c. Thus, it needs to be calculated only once for each server s and the calculation result can be reused for all unassigned clients. Moreover, its value can be maintained

incrementally across iterations. At the end of the i-th iteration, we can compute the value of $T_{i+1}^{(2)}(s)$ for the next iteration by $T_{i+1}^{(2)}(s) = T_i^{(2)}(s) + 2d(s, s_{c_i^*})$ (lines 17 to 18), where c_i^* is the client newly assigned in the i-th iteration.

Given a server s, we only need to look for the client that minimizes $T_i^{(1)}(c, s)$ in order to find the minimum increase in the total interaction path length. This client in fact must be the nearest unassigned client to the given server. To efficiently locate the nearest unassigned client to each server, our implementation pre-computes a two-dimensional table L in the pre-processing stage of the genetic algorithm, where $L[s][i]$ stores the information of the i-th nearest client $(1 \leq i \leq |C|)$ to candidate server location $s \in S$. In the greedy assignment subroutine, we maintain a pointer $p(s)$ for each server $s \in S'$ pointing to the first unassigned client in $L[s]$. In this way, we can easily pick up the nearest unassigned client of each server in any iteration by $L[s][p(s)]$. Our greedy assignment subroutine examines the nearest unassigned client of each server and selects the one that minimizes formula (2) for assignment in each iteration.

The value of total interaction path length T can be calculated by

$$\begin{aligned} T &= \sum_{i=1}^{|C|} T_i(c_i^*, s_{c_i^*}) \\ &= \sum_{i=1}^{|C|} \left(T_i^{(1)}(c_i^*, s_{c_i^*}) + T_i^{(2)}(s_{c_i^*}) \right) + \sum_{i=1}^{|C|} T_i^{(3)}. \end{aligned}$$

Note that

$$\begin{aligned} \sum_{i=1}^{|C|} T_i^{(3)} &= 2 \cdot \sum_{i=1}^{|C|} \sum_{j=1}^{i-1} d(s_{c_j^*}, c_j^*) \\ &= 2 \cdot \sum_{i=1}^{|C|} (|C| - i) \cdot d(s_{c_i^*}, c_i^*). \end{aligned}$$

Therefore, we can compute the value of $\sum_{i=1}^{|C|} T_i^{(3)}$ and T incrementally across iterations (line 16). The fitness value $f(o)$ of a chromosome o is subsequently set to the inverse of its total interaction path length calculated. The greedy assignment subroutine has a time complexity of $O(|C| \cdot |S'|) = O(|C| \cdot k)$, implying that the fitness value of a chromosome can be evaluated efficiently.

D. Crossover Operation

The objective of the crossover operation is to exchange the information contained in two parent chromosomes and generate an offspring that may outperform its parents. We adopt the roulette-wheel parent selection [7] that allows chromosomes with shorter total interaction path lengths to have higher probabilities to be selected as the parents for the crossover operation. Specifically, the probability that a chromosome o_i is selected is given by $\frac{f(o_i)}{\sum_{j=1}^{p} f(o_j)}$.

Our crossover operator works in the following way. The candidate server locations that are chosen in both parents are inherited in the offspring. For the candidate server locations that are chosen by only one of the parents, we randomly pick up some of them according to the probabilities that are proportional to the fitness of the parents holding them. More specifically, let $o_a = (g_1^{o_a}, g_2^{o_a}, ..., g_{|S|}^{o_a})$ and $o_b = (g_1^{o_b}, g_2^{o_b}, ..., g_{|S|}^{o_b})$ be the two parent chromosomes and $r = (g_1^r, g_2^r, ..., g_{|S|}^r)$ be the offspring. Let $f(o_a)$ and $f(o_b)$ denote the fitness values of o_a and o_b respectively. First, we scan the bits in the two parents and set $g_i^r = 1$ $(1 \leq i \leq |S|)$ if both $g_i^{o_a}$ and $g_i^{o_b}$ are 1. Let G_A be the set of bit indexes whose corresponding candidate server locations are chosen by o_a but not o_b, i.e., $G_A = \{i | g_i^{o_a} = 1, g_i^{o_b} = 0\}$. Similarly, let $G_B = \{i | g_i^{o_a} = 0, g_i^{o_b} = 1\}$. For each $i \in G_A$, bit g_i^r would be set to 1 with probability $\frac{f(o_a)}{M \cdot f(o_a) + M \cdot f(o_b)}$. For each $i \in G_B$, bit g_i^r would be set to 1 with probability $\frac{f(o_b)}{M \cdot f(o_a) + M \cdot f(o_b)}$. The bits of indexes in $G_A \cup G_B$ are randomly picked and set to 1 with the above probabilities until a total of k bits are set to 1 in the offspring r.

After r is generated, its fitness value is evaluated. Then, we apply the following replacement strategy to accept it. We define the similarity between two chromosomes to be the number of their common server locations. If the offspring r is more similar to o_a than to o_b, we replace o_a by r in the population only if r has a higher fitness value than o_a. If r is more similar to o_b than to o_a, we replace o_b by r only if r has a higher fitness value than o_b. Otherwise, r is eliminated and is not selected in the next generation. The advantage of such replacement is that it maintains diversity in the population and therefore prevents premature convergence.

```
1:  Initialize table L for the greedy assignment subroutine;
2:  Initialize the chromosome population;
3:  Fitness evaluation for all chromosomes in the population;
4:  Generation number E ← 1;
5:  while E ≤ a predefined maximum generation number do
6:      Select two parent chromosomes;
7:      Create an offspring;
8:      Fitness evaluation for the offspring;
9:      Replacement;
10:     Mutation operation with a mutation probability;
11:     E ← E + 1;
12: Output the best chromosome in the population;
```

Algorithm 3: Enhanced Genetic Algorithm

E. Mutation Operation

The purpose of the mutation operation is to introduce new gene materials into the population so as to prevent the genetic algorithm from premature convergence [11]. It is performed in each generation with a mutation probability. In our design, we substitute the worst chromosome in the population by a completely new one. The new chromosome is generated in the same way as that in the initialization stage, i.e., the values of bits are set according to the k server

locations selected by the greedy k-median subroutine for a subset of $\frac{1}{2}|C|$ randomly selected clients.

Our genetic algorithm terminates when a predefined maximum generation number is reached. An overview of the proposed genetic algorithm is depicted in Algorithm 3.

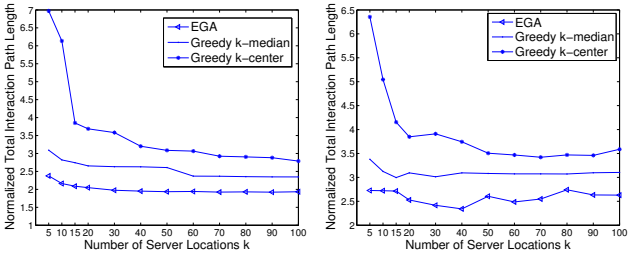
V. Experimental Evaluation

A. Experimental Setup

In this section, the proposed enhanced genetic algorithm is evaluated by simulation experiments. The network topologies used in the experiments are from three datasets: Meridian [1], RANDOM and WAXMAN. Meridian is a real world dataset containing pairwise latency measurements between 2500 nodes. Since some measurements in this dataset are missing, we randomly pick 1000 nodes that are not involved in missing measurements and put the complete pairwise latency data of these nodes to use. The RANDOM and WAXMAN datasets each contains 50 network topologies generated from the synthetic network topology generator GT-ITM [3] using the random and Waxman models [16] respectively. For each topology in these two datasets, 100 nodes are randomly placed on a plane of size 1000 distance units by 1000 distance units. In the random model, a link is established between each pair of nodes with a fixed probability. In the Waxman model, the probability of establishing a link between two nodes u and v is given by $p(u,v) = \beta e^{\frac{-d(u,v)}{l\alpha}}$, where $d(u,v)$ is the Euclidian distance between u and v, and l is the maximum Euclidian distance between any two nodes on the plane. α and β are Waxman parameters that influence the density of links. The weight of each link is given by the Euclidean distance between the endpoints of the link. The network topologies in the RANDOM and WAXMAN datasets are randomly generated with different parameter settings. For each dataset, we run the algorithms on all 50 topologies and compute the average results for comparison. We also assume that in each topology of these three datasets, a client is located at each node in the network, and all the nodes are candidate server locations.

The default population size and mutation probability for our enhanced genetic algorithm (which we shall abbreviate as EGA onwards) are set to 80 and 0.15 respectively. For the purpose of comparison, we also include a greedy k-median heuristic and a greedy k-center heuristic in the experiments. The greedy k-median algorithm computes the k-median server locations for all clients following the same design principle as that introduced in Section IV-B. The greedy k-center algorithm starts from an empty set of server locations. In each iteration, it selects a new candidate server location that yields the largest reduction of the maximum distance between all clients and their nearest servers. The algorithm stops when k server locations are selected.

After obtaining the server locations selected by the server placement algorithms, we apply two assignment strategies to assign clients to servers in our experiments. The first one is the greedy assignment strategy described in Section IV-C, while the second strategy is a nearest assignment strategy that assigns each client to its nearest server.



(a) With greedy client assignment (b) With nearest client assignment

Figure 3. Performance of the algorithms using Meridian dataset

To compare the relative performance of the algorithms, we normalize the total interaction path length produced by each algorithm by a theoretical lower bound which is derived by making two relaxations. The first relaxation is that clients could be assigned to any candidate server location (rather than just a subset of k selected locations), and the second relaxation is that a client may be assigned to different servers for interacting with different clients. Let S be the set of candidate server locations, and C be the set of clients. Then, the lower bound is given by

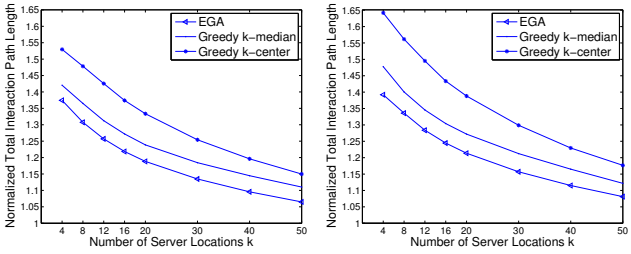
$$\sum_{c \in C} \sum_{c' \in C} \left(\min_{s,s' \in S} \left(d\left(c,s\right) + d\left(s,s'\right) + d\left(s',c'\right) \right) \right).$$

Note that due to the above relaxations, the theoretical lower bound is a super-optimum that may be unachievable.

B. Performance Comparison for Different Algorithms

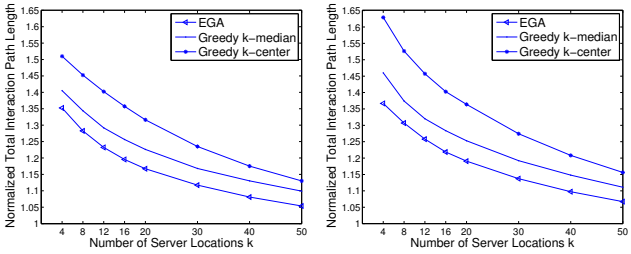
Figure 3 shows the normalized total interaction path lengths produced by the algorithms for different numbers of server locations selected using the Meridian dataset. As can be seen, EGA always outperforms the other two algorithms. For example, in Figure 3(a), the normalized total interaction path length produced by the greedy k-median algorithm with $k = 60$ is close to that of EGA with $k = 5$. In the meantime, the greedy k-center algorithm cannot achieve similar results even with 100 server locations. This implies that EGA could save a lot of server resources and provide better interactivity for the DIA compared to the greedy k-median and greedy k-center algorithms. In both Figures 3(a) and 3(b), the performance of EGA with $k = 10$ completely surpasses that of the other two algorithms with up to 100 server locations.

Figures 4 and 5 show the performance of the three algorithms with different client assignment strategies using the RANDOM and WAXMAN datasets. For both datasets, we assume that messages are routed along shortest paths in our experiments. As a result, the network latencies among nodes follow the triangle inequality. All the experimental



(a) With greedy client assignment (b) With nearest client assignment

Figure 4. Performance of the algorithms using RANDOM dataset



(a) With greedy client assignment (b) With nearest client assignment

Figure 5. Performance of the algorithms using WAXMAN dataset

results show that EGA outperforms the rest server placement algorithms regardless of the client assignment strategy used, and obtains average improvements of 4.5% and 11.2% over the greedy k-median and greedy k-center algorithms respectively for the same number of servers. In addition, it can be seen that, with the same server placement algorithm, the results obtained by the greedy client assignment are better than those obtained by the nearest client assignment. But the effect of using the greedy assignment strategy is not as significant as that of using a more efficient server placement algorithm like the EGA algorithm. It could be observed from Figures 4 and 5 that with the same number of servers k, the results of EGA with the nearest client assignment are still better than those of the rest server placement algorithms with the greedy client assignment. For example, when $k = 4$, the result of the EGA algorithm in Figure 4(b) is 1.39 and is still better than the corresponding results of the greedy k-center and the greedy k-median algorithms in Figure 4(a) which are 1.53 and 1.42 respectively. Similar trends could also be observed for other numbers of servers.

VI. Conclusion

Network latencies pose a challenge to the development of DIAs. In this paper, we aim to reduce the interaction times and improve the quality of interaction between clients in DIAs. We focus on the server placement strategy that could directly reduce the magnitude of client-server as well as inter-server latencies involved in the interaction. An enhanced genetic algorithm has been proposed and experimentally evaluated. The design of the genetic operators in the algorithm is based on the nature of the server placement problem. Experimental results using three different datasets show that our algorithm leads to appreciable improvement of the interaction quality between clients.

References

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

[2] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *ACM NetGames '04*, pages 144–151, 2004.

[3] K. Calvert and E. Zegura. GT internetwork topology models (GT-ITM). http://www.cc.gatech.edu/projects/gtitm/gt-itm/.

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

[5] 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.

[6] J. C. S. Lui and M. F. Chan. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):193–211, 2002.

[7] M. Melanie. *An introduction to genetic algorithms*. The MIT Press, 1999.

[8] L. Qiu, V. N. Padmanabhan and G. M. Voelker. On the placement of web server replicas. In *Proc. IEEE INFOCOM '01*, pages 1587–1596, 2001.

[9] L. D. Briceno, H. J. Siegel, A. A. Maciejewski, H. Ye, B. Lock, M. N. Teli, F. Wedyan, C. Panaccione and Z. Chen. Resource allocation in a client/server hybrid network for virtual world environments. In *Proc. IEEE IPDPS '08*, pages 1–13, 2008.

[10] L. Pantel and L. C. Wolf. On the suitability of dead reckoning schemes for games. In *ACM NetGames '02*, pages 79–84, 2002.

[11] M. Srinivas and L. M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994.

[12] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis and A. Bestavros. Distributed placement of service facilities in large-scale networks. In *Proc. IEEE INFOCOM '07*, pages 2144–2152, 2007.

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

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

[15] C. Bouras, A. Philopoulos, T. Tsiatsos. e-Learning through distributed virtual environments. *Journal of Network and Computer Applications*, 24(3):175–199, 2001.

[16] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[17] L. Zhang and X. Tang. Client assignment for improving interactivity in distributed interactive applications. In *Proc. IEEE INFOCOM '11*, pages 3227–3235, 2011.

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