10-2007

A two-phase approach to interactivity enhancement for large-scale distributed virtual environments

Nguyen Binh Duong TA
Singapore Management University, donta@smu.edu.sg

Suiping ZHOU

Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 4131–4152
Computer Networks
www.elsevier.com/locate/comnet

A two-phase approach to interactivity enhancement for large-scale distributed virtual environments

Duong Nguyen Binh Ta *, Suiping Zhou

School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore

Received 7 June 2006; received in revised form 30 January 2007; accepted 1 May 2007
Available online 22 May 2007

Responsible Editor: L.G. Xue

Abstract

Distributed virtual environments (DVEs) are distributed systems that allow multiple geographically distributed clients (users) to interact simultaneously in a computer-generated, shared virtual world. Applications of DVEs can be seen in many areas nowadays, such as online games, military simulations, collaborative designs, etc. To support large-scale DVEs with real-time interactions among thousands or even more distributed clients, a geographically distributed server architecture (GDSA) is generally needed, and the virtual world can be partitioned into many distinct zones to distribute the load among the servers. Due to the geographic distributions of clients and servers in such architectures, it is essential to efficiently assign the participating clients to servers to enhance users' experience in interacting within the DVE. This problem is termed the client assignment problem (CAP) in this paper. We propose a two-phase approach, consisting of an initial assignment phase and a refined assignment phase to address the CAP. Both phases are shown to be NP-hard. Several heuristic assignment algorithms are then devised and evaluated via extensive simulations with realistic settings. We find that, even under heterogeneous environments like the Internet where accurate input data for the assignment algorithms are usually impractical to obtain, the proposed algorithms are still beneficial to the performances of DVE.

Keywords: Distributed virtual environments; Geographically distributed server architecture; Client assignment; Interactivity enhancement

1. Introduction

In recent years, advances in high-speed networking technologies, computer graphics and CPU processing power have enabled the rapid development of distributed virtual environments (DVEs). DVEs are distributed systems that allow multiple geographically distributed clients (users) to explore and interact with each other in real-time within a shared, computer-generated 3D virtual world [1], where each client is represented by an *avatar*. A client controls the behavior of his/her avatar by various *inputs*, and the *updates* of an avatar's state need to be sent to other clients in the same *zone* of the virtual world to support the interactions among clients. DVEs have been applied in many areas, such as collaborative design, military simulations, e-learning and multiplayer games [1].

* Corresponding author.
E-mail addresses: binhduong@pmail.ntu.edu.sg (D.N. Binh Ta), asspzhou@ntu.edu.sg (S. Zhou).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.05.002

Developing DVEs faces many challenges. In particular, various resources are needed, e.g., network bandwidth, CPU cycles, etc. The resource requirements of DVEs may increase quickly as the number of simultaneous clients increases. In addition, the network latency may damage the interactivity and consistency of DVEs [2]. Moreover, since DVEs are human-in-the-loop applications, cheating is also an important problem that needs to be considered. Thus, usually a server-based communication architecture is employed for DVE applications. For example, popular Massively Multiplayer Online Games (MMOGs) such as Everquest [3] and Ultima Online [4] are operating on large clusters of servers. However, putting all servers at a central geographic location may result in high communication delays for clients which are far from the servers. Moreover, this centralized architecture is not scalable, since the Internet connection for the whole server cluster may fail. Someone may argue that multihoming[1] can resolve this problem. However, we should note that in multihoming, two independent network links from different ISPs may actually share the same transmission line [5]. This problem essentially forms a single point of failure in the network connections.

Therefore, a *geographically distributed server architecture* (GDSA) is desirable to support large-scale DVEs [6,7]. With this architecture, multiple geographically distributed servers are connected to each other, usually via well-provisioned connections. Each client is connected to one of these servers, and clients interact with each other through these servers.

In order to deal with large-scale DVEs with hundreds, or even thousands of clients interacting simultaneously, usually the virtual world is spatially partitioned into several distinct *zones*, with each zone managed by only one server, as in [3]. A client only interacts with other clients in the same zone,[2] and may move to other zones. As a server only needs to handle one or more zones instead of the entire virtual world, the system is more scalable. In this paper, we refer to such a partitioning approach as the *zone-based approach*.

[1] Multihoming is a networking technique to enhance the reliability of the Internet connection of an IP network. In this technique, the network in question establishes connections to two or more completely different ISPs [5].

[2] For simplicity, we say that a client is in a zone if its avatar is currently residing in that zone.

Due to the fact that clients in a DVE are geographically distributed and the heterogeneous nature of the Internet, a bad assignment of clients to servers would result in degraded interactivity for the DVE. For example, one of the most popular approach to assign clients to servers in a DVE is only based on the resource limitation, hence the name *resource-driven distribution* [8,9]. Since the network delays are not taken into account in this approach, it is very likely that a large number of clients in the DVE may have poor interactivity due to large client–server network delays. Thus, there is a strong need for efficient mechanisms to assign the participating clients to servers to reduce the client–server communication delay. This is referred to as the *client assignment* problem (CAP) in this paper. We propose a two-phase approach to the CAP. In the *initial assignment* phase, the zones of the virtual world are assigned to servers. Then, in the *refined assignment* phase, a client is assigned to an appropriate (usually nearby) server to communicate with the server that is hosting the client's zone. Based on this two-phase approach, several assignment algorithms are devised and evaluated. We show that the greedy algorithms that take into account the client–server round-trip network delays significantly outperform a delay-oblivious algorithm, even when the input data, i.e., network delays, to these algorithms are inaccurate.

The rest of the paper is organized as follows. Section 2 describes the geographically distributed server architecture, the client assignment problem and some related work. The proposed algorithms are presented in Section 3. Section 4 addresses some practical considerations in implementing the assignment algorithms. Section 5 describes our evaluation methodology, and results are discussed in Section 6. Section 7 concludes the paper.

2. Problem formulation

2.1. Notations and concepts

In this paper, we focus on DVEs that adopt the geographically distributed server architecture (GDSA) [6,7] and the zone-based partitioning approach as shown in Fig. 1. All geographically distributed servers are usually fully meshed with well-provisioned network connections, although this is not a required assumption for our approach.

Before formulating the client assignment problem, we introduce the following notations:

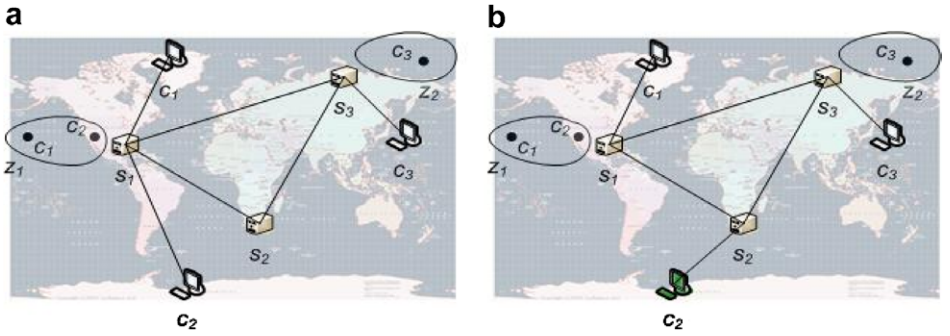

Fig. 1. Geographically distributed server architecture (GDSA).

- c_i – A client in the DVE.
- $C = \{c_1, \ldots, c_k\}$ – The set that consists of all clients in the DVE.
- z_i – A zone in the DVE.
- $Z = \{z_1, \ldots, z_n\}$ – The set that consists of all zones in the DVE.
- s_i – A server in the DVE.
- $S = \{s_1, \ldots, s_m\}$ – The set that consists of all servers in the DVE. We now introduce the following definitions:

Definition 2.1 (*Contact server*). The *contact server* of a client is the server that the client directly connects to. A client only sends inputs to its contact server. The contact server may process the inputs and send updates to the client if it is hosting the client's zone, or it may forward the client's inputs to another server which is hosting the client's zone. We denote the contact server of a client c_j as $s^c(c_j)$.

For example, in Fig. 1a, s_1 is the contact server of c_1 and c_2.

Definition 2.2 (*Target server*). The *target server* of a client is the server that is hosting the client's zone. Inputs from a client will be forwarded to its target server. The target server may send updates to the client directly if it is also the contact server of the client, or it may send to the client indirectly via the client's contact server. All clients in a zone have the same target server (therefore, we may say "the target server of a zone"), while they may have different contact servers. We denote the target server s_i of a client c_j or a zone z_k as $s^t(c_j)$ or $s^t(z_k)$, respectively.

For example, in Fig. 1a, s_1 is both the contact and target server of c_1 and c_2, i.e., $s^c(c_1) = s^c(c_2) = s^t(c_1) = s^t(c_2) = s_1$. In Fig. 1b, we switch c_2 to server s_2 (but the avatar of c_2 is still in zone z_1), the target server of c_1 and c_2 is still s_1, the contact server of c_1 is s_1, while the contact server of c_2 is now s_2. Inputs from c_2 are forwarded to s_1 by s_2. Since the network connection between s_1 and s_2 is well-provisioned with little congestion, the communication between c_2 and s_1 may now have shorter delay (assuming s_2 is closer to c_2 than s_1). However, this incurs extra resource utilization for inter-server communication between s_1 and s_2.

- R_{s_i} – The resource consumption on a server s_i. This can be measured by CPU usage, network bandwidth usage, etc. Since the network bandwidth often represents the major operating cost in current server-based MMOGs [10], in this paper, we assume that the server CPU is not a bottleneck, and measure the resource consumption by the network bandwidth usage only.
- $R^{\mathrm{T}}_{c_i}$ – The amount of resource (network bandwidth) utilized by a client c_i on its target server. Note that $R^{\mathrm{T}}_{c_i} > 0, \forall c_i$.
- $R^{\mathrm{C}}_{c_i}$ – The amount of resource utilized by a client c_i on its contact server. Note that $R^{\mathrm{C}}_{c_i} = 0$ if the contact server and target server of c_i are the same, otherwise $R^{\mathrm{C}}_{c_i} = 2R^{\mathrm{T}}_{c_i}$, since all communications between c_i and its target server are forwarded by its contact server (assuming the resource utilization is measured by bandwidth requirement).
- R_{z_i} – The amount of resource utilized by a zone z_i on its target server. We have $R_{z_i} = \sum_{c_j \in z_i} R^{\mathrm{T}}_{c_j}$.
- C_{s_i} – The resource capacity of a server s_i.
- $d_{c_i s_j}$ – The round-trip network delay between a client c_i and a server s_j.
- D – The *delay bound* of a DVE. The delay bound indicates the required upper bound of the round-trip communication delay between a client and its target server to guarantee the interactivity of the

DVE. For different types of DVEs, there are different delay bound requirements. For example, multiplayer real-time strategy (RTS) games like Microsoft's *Age of Empires* [11] typically require a latency of 500 ms [12], while FPS games and car-racing games have much more stringent latency requirements, about 250 ms [13] and 100 ms [14], respectively. It should be noted that the communication delay between a client and its target server is different from the network delay between the client and its target server. The communication delay is the sum of the network delay and the processing delay at the target server. However, we assume that the processing delay of a client's request will not increase greatly as long as the workload of the server does not exceed a predefined threshold. Hence, in this paper the client–server communication delay is determined by the client–server network delay. In the following, the term "network delay" and "communication delay" are used interchangeably.

For interactive applications like DVEs, the client–server communication delay is the most important *Quality of Service* (QoS) parameter that the system provides to clients [13]. In this paper, we say that a client is *with QoS* or *without QoS* if the communication delay between the client and its target server is smaller or larger than the delay bound, respectively.

2.2. The client assignment problem

Our client assignment problem (CAP) under consideration is for the static case, i.e., for a snapshot of the system, and the client-to-zone relation at this snapshot is known. With the geographically distributed server architecture and the zone-based approach, the client assignment problem (CAP) concerns how to assign the participating clients in a DVE to servers so that the total number of clients with QoS is maximized. We formulate the client assignment problem as follows.

Definition 2.3 (*Client assignment problem* (*CAP*)). For each client c_j in the DVE, find the target server s_k and the contact server s_l for c_j to maximize

$$|\{c_j \in z_i : d_{c_j s_k} = (d_{c_j s_l} + d_{s_l s_k}) \leqslant D,\ \forall z_i \in Z\}| \qquad (1)$$

subject to

$$R_{s_i} \leqslant C_{s_i}, \quad \forall s_i \in S, \qquad (2)$$

where $|\cdot|$ denotes the cardinality of a set.

Remark 2.1. In Definition 2.3, if s_l and s_k refer to the same server, then $d_{s_l s_k} = 0$.

Remark 2.2. The constraint (2) ensures that the resource consumption on each server s_i will not exceed its capacity.

2.3. A two-phase approach to the CAP

With the GDSA and the zone-based partitioning approach, we notice that a client may need to connect to a different server from its target server to minimize the client–server communication delay. In fact, direct routing in the Internet may result in a larger end-to-end delay than indirect routing, since the network congestion level in the links along the direct path may be higher than that in the indirect path. Several existing researches had leveraged this fact to reduce the end-to-end network delay using indirect routing, for example Resilient Overlay Network [15] and Detour [16]. Here, if we assume that the links between each pair of servers are well-provisioned with high bandwidth capacities, an indirect routing via these links is very likely to reduce the client–server communication delay.

Therefore, to address the client assignment problem, in this paper we propose a two-phase approach as follows. First, in the *initial assignment* phase, we assign the zones to servers, i.e., find a target server for each client. Then, in the *refined assignment* phase, each client is assigned to an appropriate server to communicate with its target server, i.e., find a contact server for each client. As mentioned above, if the network links between servers are well-provisioned, for each client we may find a suitable contact server that is different from that client's target server to minimize the client–target server communication delay via indirect routing.

In fact, both the initial assignment and refined assignment are themselves complex optimization problems. In order to obtain good solutions to the CAP, we must optimize some "assignment costs" in both phases. In the following sections, we separately formulate the initial assignment and refined assignment problem, and prove them to be NP-hard.

2.3.1. The initial assignment problem (IAP)

To formulate the initial assignment problem, we propose the following metric to measure the cost of assigning a zone z_j to a server s_i

$$C_{ij}^{I} = |\{c_k \in z_j : d_{c_k s_i} > D\}|. \qquad (3)$$

C^I_{ij} measures the number of clients in a zone z_j that *do not* satisfy the delay bound D, i.e., without QoS. Therefore, by minimizing the total cost when all zones are assigned, the total number of clients with QoS in the DVE would be maximized. The initial assignment problem is formulated as follows.

Definition 2.4 (*Initial assignment problem* (*IAP*)). Let $I = \{1,\ldots,m\}$ and $J = \{1,\ldots,n\}$ be the set of indexes of servers and zones in the system, respectively. For each $i \in I$ and $j \in J$, given the cost C^I_{ij} of assigning zone z_j to server s_i as defined in (3), find an assignment matrix $X = (x_{ij})$, with $x_{ij} = 1$ if zone z_j is assigned to server s_i or $x_{ij} = 0$ otherwise, which minimizes the total cost

$$C^I(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} C^I_{ij}x_{ij} \tag{4}$$

subject to

$$\sum_{j=1}^{n} R_{z_j}x_{ij} \leqslant C_{s_i}, \quad \forall i \in I, \tag{5}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in J, \tag{6}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in I,\ \forall j \in J. \tag{7}$$

Remark 2.3. In Definition 2.4, constraint (5) ensures that the capacity of each server will not be exceeded. Constraint (6) means that each zone is assigned to only one server.

Remark 2.4. In the IAP, the number of binary decision variables is mn, while the number of constraints is $m + n$, i.e., $O(n)$, since the number of zones n is usually much larger than the number of servers m.

Theorem 2.1. *The IAP is NP-hard.*

Proof. We consider in our model the special case where the resource capacity C_{s_i} of each server is the same. We further assume that the cost C^I_{ij} when assigning a zone z_j, $j \in J = \{1,\ldots,n\}$ to server s_i, $i \in I = \{1,\ldots,m\}$ has the special form of Δ^{i-1}, where $\Delta > 1$ is a constant, i.e., assigning any zone to server s_1 costs $C^I_{1j} = 1$, to server s_2 costs $C^I_{2j} = \Delta$, to server s_3 costs $C^I_{3j} = \Delta^2$, etc. Then the optimization goal of IAP in this special case would be to assign all the zones to the smallest possible number of M servers, with server indexes ranging from 1 to M. This is exactly the well-known Bin Packing problem [17], in which a bin corresponds to a server and an item corresponds to a zone. Since IAP generalizes the Bin Packing problem, which is NP-hard, IAP is also NP-hard.[3] □

2.3.2. The refined assignment problem (RAP)

After the initial assignment phase, some clients may still be without QoS. The refined assignment phase will attempt to further increase the number of clients with QoS in the DVE. One possible approach is to exploit the well-provisioned inter-server connections, if they are available. We propose the following metric for the refined assignment to measure the cost of selecting server s_k as the contact server for a client c_j, where s_i is c_j's target server

$$C^R_{ij} = \begin{cases} (d_{c_j s_k} + d_{s_k s_i}) - D, & \text{if } (d_{c_j s_k} + d_{s_k s_i}) > D, \\ 0, & \text{if } (d_{c_j s_k} + d_{s_k s_i}) \leqslant D. \end{cases} \tag{8}$$

C^R_{ij} measures the "distance" from the delay bound D of the communication delay of a client c_j if c_j is assigned to server s_k as its contact server. Therefore, by minimizing the total cost when all clients are assigned, the total number of clients with QoS in the DVE would be maximized. The refined assignment problem is then formulated as follows.

Definition 2.5 (*Refined assignment problem* (*RAP*)). Let $I = \{1,\ldots,m\}$ and $J = \{1,\ldots,k\}$ be the set of indexes of servers and clients in the system, respectively. For each $i \in I$ and $j \in J$, given the cost C^R_{ij} of selecting server s_i as the contact server of client c_j, find an assignment matrix $X = (x_{ij})$, with $x_{ij} = 1$ if client c_j takes server s_i as its contact server or $x_{ij} = 0$ otherwise, which minimizes the total cost

$$C^R(X) = \sum_{i=1}^{m}\sum_{j=1}^{k} C^R_{ij}x_{ij} \tag{9}$$

subject to

$$\sum_{j=1}^{k} R^{\mathrm{C}}_{c_j}x_{ij} \leqslant \left(C_{s_i} - \sum_{c_k} R^{\mathrm{T}}_{c_k}\right), \ s_i = s'(c_k), \forall i \in I, \tag{10}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \ \forall j \in J, \tag{11}$$

$$x_{ij} \in \{0,1\}, \ \forall i \in I, \forall j \in J. \tag{12}$$

[3] We would like to thank Gunther Raidl for helping with the NP-hardness of the Generalized Assignment Problem and hence this proof.

Remark 2.5. In Definition 2.5, constraint (10) ensures that the *remaining* capacity of each server, i.e., the amount of resource of each server that still remains after the initial assignment, will not be exceeded. Constraint (11) means that each client is assigned to only one contact server.

Remark 2.6. In the RAP, the number of binary decision variables is mk, while the number of constraints is $m + k$, i.e., $O(k)$, since the number of clients k is usually much larger than the number of servers m.

Theorem 2.2. *The RAP is NP-hard.*

Proof. The proof is similar to that for the IAP. □

2.4. Related work

To the best of our knowledge, there is no existing work that directly addresses the client assignment problem in DVEs as we have described. Research on how to assign clients to servers in DVEs is usually formulated as a load distribution problem in a locally distributed server architecture, i.e., all the servers are placed in the same machine room [18,19]. Such approaches may damage the interactivity of the DVE, since clients may be far away (in terms of network delays) from the servers.

Recently, the interest in investigating server and network architectures to reduce the effect of network latency for DVEs has been increased [20,8,9]. In [8,9], the authors introduced the concept of *latency driven distribution* (LDD) for the distribution of a DVE over the networking architecture, as opposed to the traditional *resource driven distribution* (RDD), i.e., load distribution. Our client assignment problem shares the idea of the LDD concept. However, the authors of [8,9] only investigate the provision of immersive audio communication in DVEs within the context of the LDD concept. In their work, two clients that belong to two adjacent DVE zones are able to hear each other. However, several important kinds of interactions that may happen in a DVE such as shooting enemies, manipulating objects, changing object ownership, etc., were not studied. These interactions may raise complicated consistency issues, if the clients in adjacent zones are managed by different physical server machines.

In a more recent work [21], the authors proposed a distributed algorithm for clients to selects the best server in a mirrored architecture for online games, taking into account the network delay between clients and servers. The mirrored architecture replicates the DVE zones at multiple servers. This approach shares some similarities with the web server replica placement problem in CDNs [22,23]. However, unlike the replication of web documents, DVE replication faces serious consistency problems [2] which may damage the users' experience in interacting with the virtual world. In our approach, which employs the discrete partitioning strategy, only one server has the control over the state of a zone, thus consistency can be enforced.

3. Client assignment algorithms

To address the client assignment problem (CAP), in this section we propose some algorithms for the initial assignment (IAP) and then for the refined assignment (RAP). Since both IAP and RAP are NP-hard, we seek heuristic solutions instead of optimal ones. We start with algorithms for the IAP, with which we assign zones to servers, i.e., determine the target servers for clients. We propose three algorithms for the IAP: the first one randomly assigns the zones, while the second one is a greedy heuristics to minimize the number of clients without QoS in the system. The third algorithm for IAP differs from the second one only in the cost metric used.

After the target servers for clients are determined, in the refined assignment phase, we find an appropriate contact server for each client. This assignment phase depends on the quality, i.e., network latency, of the inter-server connections. If the inter-server connections are not well-provisioned, i.e., the servers use the commodity Internet to communicate with each other, the only feasible refined assignment approach is to connect each client to the server hosting its zone, i.e., based on the virtual location of clients. On the other hand, if servers are fully meshed with good network connections, e.g., private, dedicated networks, or QoS-enable networks, then it is possible to reduce the client–server communication delay of a client by assigning that client to a contact server different from its target server. In this case, we propose two algorithms. The first algorithm selects the closest server of a client as its contact server, and the second algorithm is a

greedy heuristics similar to the one for solving the IAP.

Finally, by combining the algorithms for the IAP and the RAP, we have the two-phase algorithms for the CAP. For comparison purpose, we also present the optimal algorithm using the "branch-and-bound" method to solve the integer programming formulation of IAP and RAP.

3.1. Algorithms for the IAP

3.1.1. Random assignment of zones (RanZ)

In RanZ, zones are assigned to randomly selected servers with the only concern of not overloading the servers. The pseudo-code of the algorithm is shown in Fig. 2. In this algorithm, the following procedure is repeated until all zones have been assigned: first we randomly select a zone z_j, and then a server s_i with sufficient capacity is randomly selected to take z_j, i.e., the target server of all clients in z_j is set to s_i.

Remark 3.1. The computation cost of the RanZ algorithm is O(mn), where n is the number of zones and m is the number of servers.

Remark 3.2. The communication cost of the RanZ algorithm is O(m), where m is the number of servers in the system, since the master server that runs RanZ needs to contact m servers to obtain the workload information.

3.1.2. Greedy assignment of zones – algorithm 1 (GreZ-1)

Since RanZ is oblivious to client–server network delays when assigning zones to servers, the obtained performance in terms of the number of clients with QoS may not be good, i.e., the cost of the assignment as defined in Eq. (4) may be high. Hence, in the GreZ-1 algorithm, we use a greedy heuristics to minimize the total number of clients *without* QoS in the system.

The pseudo-code is shown in Fig. 3. Let $\mu_{ij} = -C^I_{ij}$ (recall that C^I_{ij} measures the number of clients without QoS in zone z_j if z_j is assigned to s_i) be a heuristic measure of the *desirability* of assigning zone z_j to server s_i. The smaller the cost C^I_{ij} is, the higher the desirability μ_{ij} is. The algorithm iteratively considers all the unassigned zones and pick a zone z_j with the maximum difference ρ_j between the largest desirability $\mu_{i_j j}$ and the second largest desirability μ_{sj}. Then, z_j is assigned to a server s_i with the highest value of μ_{ij} and with sufficient resource capacity. This procedure is adapted from the well-known approach used to solve the Generalized Assignment Problem [24,25].

Let us illustrate the effectiveness of the GreZ-1 algorithm using an example. Assuming that the DVE has two zone z_1 and z_2, and two servers s_1 and s_2. Each server can only take one zone, due to its capacity constraint. Let us further assume that assigning z_1 to s_1 or s_2 costs $C^I_{11} = 10$ or $C^I_{21} = 20$, respectively. Similarly, assigning z_2 to s_1 or s_2 costs $C^I_{12} = 5$ or $C^I_{22} = 10$, respectively.

The desirability difference ρ_1 of zone z_1 is equal to $\mu_{11} - \mu_{21} = -C^I_{11} - (-C^I_{21}) = -10 - (-20) = 10$, while ρ_2 of z_2 is equal to $\mu_{12} - \mu_{22} = -C^I_{12} - (-C^I_{22}) = -5 - (-10) = 5$. Hence, the GreZ-1 algorithm will select z_1 to assign first. As a result, z_1 is assigned to server s_1, and z_2 is assigned to server s_2, since each server can only take one zone. The

```
1 begin
2     while Z ≠ ∅ do
3         randomly select a zone z_j from Z;
4         randomly select a server s_i, with R_{s_i} + R_{z_j} ≤ C_{s_i};
5         assign z_j to s_i;
6         R_{s_i} = R_{s_i} + R_{z_j};
7         remove z_j from Z
8     end
9 end
```

Fig. 2. IAP – Random assignment of zones.

```
1  begin
2      foreach c_k ∈ C do
3          foreach s_i ∈ S do
4              if d_{c_k s_i} > D then
5                  C^I_ij = C^I_ij + 1, c_k ∈ z_j;
6              end
7          end
8      end
9      foreach z_j ∈ Z do
10         sort the list L^M_j of values μ_ij = −C^I_ij in descending order, s_i ∈ S;
11         find ρ_j = μ_{i_j j} − max_{s≠i_j} μ_sj, where i_j = arg max_i μ_ij;
12     end
13     sort the list of zones Z according to the value ρ_j of each zone z_j in descending order;
14     while Z ≠ ∅ do
15         pick the first zone z_j ∈ Z;
16         while L^M_j ≠ ∅ do
17             pick the first desirability μ_ij ∈ L^M_j;
18             pick server s_i;
19             if R_{s_i} + R_{z_j} ≤ C_{s_i} then
20                 assign z_j to s_i;
21                 R_{s_i} = R_{s_i} + R_{z_j};
22                 remove z_j from Z;
23                 break;
24             end
25             remove μ_ij from L^M_j;
26         end
27     end
28 end
```

Fig. 3. IAP – Greedy assignment of zones – algorithm 1.

total cost of this assignment is $10 + 10 = 20$, i.e., in total there are 20 clients that are without QoS in the system.

However, if we do not follow the procedure of GreZ-1, then z_2 may be chosen to assign first. In this case, z_2 will be assigned to s_1, thus z_1 has to be assigned to s_2. The total cost in this case would be $20 + 5 = 25$, which is larger than the cost obtained using GreZ-1. Thus, by sorting the zones according to their desirability differences ρ, GreZ-1 can effectively reduce the number of clients without QoS in the system.

3.1.2.1. Complexity analysis. We first analyze the computation cost of the GreZ-1 algorithm. Let m, n and k denote the number of servers, the number

of zones and the total number of clients, respectively. The GreZ-1 algorithm consists of four parts. The first part (from line 2 to line 8) is to find all the values μ_{ij}. The cost of this part is $O(mk)$. The second part (from line 9 to line 12) is to find ρ_j for all $z_j \in Z$. In the line 10, Quicksort is used. Hence, the cost of this part is $O(n)[O(m\log m) + O(1)]$, or $O(mn\log m)$.

The third part (line 13) is to sort the list of n values ρ_j. Using Quicksort, the cost is $O(n\log n)$. The final part (from line 14 to line 27) is to assign zones to servers. There are m servers and n zones, hence the cost of this part is $O(nm)$.

Hence, the overall computational cost of the GreZ-1 algorithm is $O(mk) + O(mn\log m) + O(n\log n) + O(nm)$, or $O(\max\{O(mk), O(mn\log m), O(n\log n)\})$.

Remark 3.3. The computation complexity of the GreZ-1 algorithm is $O(\max\{O(mk), O(mn\log m), O(n\log n)\})$, where n is the number of zones, m is the number of servers and k is the number of clients in the system. In practice, usually $k \gg m$ and $k \gg n$, hence the overall complexity of GreZ-1 for most cases can be written as $O(mk)$.

We now analyze the communication cost of GreZ-1. In this algorithm, the master server needs to obtain the network delays for each pair of client and server in the system using network measurement tools like King [26]. The communication cost of this operation is $O(mk)$. Similar to the case of RanZ, the master server also needs to obtain workload information from m servers. Hence, the total communication cost of GreZ-1 is $O(mk + m)$, or $O(mk)$.

Remark 3.4. The communication complexity of the GreZ-1 algorithm is $O(mk)$, where m is the number of servers and k is the number of clients in the system.

3.1.3. Greedy assignment of zones – algorithm 2 (GreZ-2)

The only difference between GreZ-1 and GreZ-2 lies in the cost metric they use. With the cost metric in Eq. (3), Greedy-1 aims to minimize the total number of clients without QoS in the system directly. For the GreZ-2 algorithm, we propose to use the following metric to measure the cost when assigning a zone z_j to a server s_i

$$C'_{ij} = \frac{\sum_{c_k \in z_j} d_{c_k s_i}}{|z_j|}. \quad (13)$$

C'_{ij} measures the average delay from all client $c_k \in z_j$ to server s_i if they are all assigned to s_i. By minimizing this cost function, it is expected that the selected server s_i for z_j will be near (in terms of network delay) to the center of mass of the client population of z_i, thus the number of clients with QoS may become large.

3.2. Algorithms for the RAP

3.2.1. Normal inter-server networks

In this case, we assume that the inter-server networks are not well-provisioned, i.e., the communication between any two servers in the system is done via the commodity Internet (hence the name "normal" inter-server networks). Hence, there is no incentive to forward messages from a client to its target server via another contact server. We propose an algorithm, referred to as the *Virtual Location based assignment of clients*, for the RAP.

3.2.1.1. Virtual location based assignment of clients (VirC). The VirC algorithm for the RAP adopts the most "natural" way to assign clients to servers in DVEs. It only considers the virtual location of each client c_j when determining a contact server for c_j, thus c_j will connect to the same server that is also hosting c_j's zone, i.e., the contact server of c_j is the same as its target server. This approach will not incur any inter-server communication cost. However, the number of clients with QoS is not improved compared to the initial assignment.

We analyze the communication cost of VirC. In this algorithm, the master server only needs to inform k clients and m servers in the system about the assignment result, thus the total communication cost is $O(k + m)$, or $O(k)$ since $k \gg m$.

Remark 3.5. The communication complexity of the VirC algorithm is $O(k)$, where k is the number of clients in the system.

3.2.2. Well-provisioned inter-server networks

In this case, we assume that the inter-server networks are well-provisioned with low network delays. We propose two algorithms for the RAP, namely the *Closest-server assignment of clients*, and the *Greedy assignment of clients*.

3.2.2.1. Closest-server assignment of clients (CloC). In the CloC algorithm, the closest server of each

client is chosen as its contact server. The main motivation of this algorithm is to reduce the client–target server communication delay by exploiting the well-provisioned inter-server networks which have low delays.

Remark 3.6. The computation complexity of the CloC algorithm is $O(km)$, where k is the number of clients and m is the number of servers in the system.

Remark 3.7. Similar to VirC, the communication complexity of the CloC algorithm is $O(k)$, where k is the number of clients in the system.

3.2.2.2. Greedy assignment of clients (GreC). The GreC algorithm is a greedy heuristics which takes into account network delay from a client to its target server when selecting contact server for the client. The pseudo-code is shown in Fig. 4. This algorithm is similar to the GreZ algorithm for the IAP. The major difference between the two algorithms lies in the cost metric used. GreC uses the cost metric defined in Eq. (8).

GreC starts by considering the round-trip delay to target server of each client in the system. If the network delay from a client c_j to its target server s_i is less than the delay bound, the algorithm selects the same server s_i as the contact server of c_j. Otherwise, c_j is added to a list L^E, which only contains clients having network delay larger than the delay bound. The next part of the algorithm attempts to assign each client in L^E to a contact server to increase the number of clients with QoS in the system. This part is similar to the GreZ algorithm, except for the cost function used.

Complexity analysis. We analyze the computational cost of GreC algorithm. Let m, k denote the number of servers and the total number of clients, respectively. The GreC algorithm consists of four parts. The first part (lines 2–9) is to construct a list of clients that are without QoS and assign contact servers for clients with QoS. The cost of this part is $O(k)$. The second part (lines 10–14) is to construct the list of values ρ_j for all clients that are without QoS. The cost of this part is $O(k)[O(m) + O(m\log m) + O(1)]$ or $O(km\log m)$. The third part (line 15) costs $O(k\log k)$ (using Quicksort). The final part (lines 16–29) is to assign clients to contact servers. The cost of this part is $O(km)$.

Hence the overall computational cost of the GreC algorithm is $O(k) + O(km\log m) + O(k\log k) + O(km)$ or $O(\max\{m\log m, \log k\}k)$.

Remark 3.8. The complexity of the GreC algorithm is $O(\max\{m\log m, \log k\}k)$, where m is the number of servers and k is the number of clients in the system.

We analyze the communication cost of GreC. In this case, the master server needs to obtain the inter-server network delays for all m servers in the system. The communication cost for this operation is $O(m^2)$. Then, it needs to inform k clients and m servers about the assignment result. Thus, the total communication cost for GreC would be $O(m^2 + k + m)$, or $O(\max\{m^2, k\})$.

Remark 3.9. The communication complexity of the GreC algorithm is $O(\max\{m^2, k\})$, where m is the number of servers and k is the number of clients in the system.

3.3. Two-phase algorithms for the CAP

A two-phase algorithm for the CAP is obtained by combining the algorithms for the IAP and the RAP. If the inter-server networks are not well-provisioned, we have the following three two-phase algorithms:

- RanZ/VirC
- GreZ-1/VirC
- GreZ-2/VirC

If the inter-server networks are well-provisioned, we have the following six different two-phase algorithms:

- RanZ/CloC
- RanZ/GreC
- GreZ-1/CloC
- GreZ-1/GreC
- GreZ-2/CloC
- GreZ-2/GreC

3.4. Optimal algorithm using the branch-and-bound method

For comparison purposes, based on the Integer Programming formulation of the IAP and RAP, we use the so-called ''branch-and-bound'' method [27] implemented in the free Mixed Integer Linear Programming (MILP) solver lp_solve [28] to obtain the optimal solutions to the CAP. lp_solve is a free MILP solver with full source code, examples and

```
1  begin
2      foreach c_j ∈ C do
3          get target server s_i of c_j;
4          if d_{c_j s_i} ≤ D then
5              select s_i as the contact server of c_j;
6          else
7              add c_j to the list L^E;
8          end
9      end
10     foreach c_j ∈ L^E do
11         find μ_ij = −C^R_ij, ∀s_i ∈ S;
12         sort the list L^M_j of values μ_ij descendingly;
13         find ρ_j = μ_{i_j j} − max_{s≠i_j} μ_sj, where i_j = arg max_i μ_ij;
14     end
15     sort the list L^R of values ρ_j in descending order;
16     while L^E ≠ ∅ do
17         pick client c_j with highest ρ_j;
18         while L^M_j ≠ ∅ do
19             pick the highest desirability μ_ij ∈ L^M_j;
20             pick server s_i;
21             if R_{s_i} + R^C_{c_j} ≤ C_{s_i} then
22                 select s_i as the contact server of c_j;
23                 R_{s_i} = R_{s_i} + R^C_{c_j};
24                 remove c_j from L^E;
25                 break;
26             end
27             remove μ_ij from L^M_j;
28         end
29     end
30 end
```

Fig. 4. RAP – Greedy assignment of clients.

detailed manuals. It can solve pure linear, mixed integer/binary, semi-continuous and special ordered sets models.

We should note that this approach is only applicable when the system size is small, otherwise the running time of lp_solve will become very long (on the order of several hours), which is clearly impractical for interactive applications like DVEs which require prompt assignment decisions.

4. Practical considerations

In this section we address some practical considerations in implementing the proposed assignment algorithms. These considerations include how to obtain input data for the algorithms and how to deal with the inherent dynamics of DVEs.

4.1. Obtaining input data

The first issue is how to obtain input data for the proposed assignment algorithms. Our algorithms are centralized, hence we need a dedicated server[4] to collect the data and run the algorithms. The input data includes the client–server and inter-server round-trip network delays, and the resource requirement of each client on its server.

4.1.1. Network delays

The server that runs our algorithms can obtain the client–server network delays using some well-known network measurement methods such as IDMaps [29] and King [26]. Both of these methods are scalable and incur little measurement overhead.

4.1.1.1. Internet Distance Map Service (IDMaps). IDMaps is a scalable Internet-wide architecture for measuring and disseminating distance information on the Internet. It was designed to be the underlying service that provides the necessary information used by SONAR/HOPS [30], which is a simple protocol for a host to quickly and efficiently learns the network distance between any two hosts. IDMaps relies on special end hosts called *tracers* deployed at some strategic locations in the Internet. Each tracer measures the distance between itself and a set of Internet hosts that are close to it. The measured distances are then sent to a set of dedicated servers, called SONAR/HOPS servers.

To estimate the distance between any two Internet hosts, the SONAR/HOPS servers calculate the sum of the distances from each host to its nearest tracer, and the distance between the two corresponding tracers. Obviously, the accuracy of such approach depends on the number and Internet locations of tracers. The goal of IDMaps is to achieve an estimation accuracy "within a factor of 2 with very high probability and often better than that" [29]. Any client can then query a SONAR/HOPS server to learn the distance between any two other hosts using client–server query/reply protocols similar to the widely used DNS query/reply protocol.

4.1.1.2. King. King [26] is a very clever approach to scalable network latency measurements. It does not require any additional infrastructure like IDMaps. Instead, King uses existing recursive DNS queries to accurately measure round-trip network delays between arbitrary Internet end hosts.

Given a pair of hosts, King first finds their two nearby DNS servers, for example a and b. It is noted that Internet hosts are usually close their respective DNS servers. By issuing a recursive DNS query to one of these name servers, for example the server a, King is able to measure a latency value which is the latency between a and b plus the latency between the machine that is running it,[5] says c, and the name server a. Then, it measures the latency between c, and the name server a using an ICMP ping. After subtracting the first latency value by the second one, King can produce a rather accurate estimation of latency between the two given hosts.

We should note that since King uses direct measurements, its accuracy may be significantly better than extrapolation-based approaches like IDMaps. In fact, in their evaluation, the authors of King claimed that their tool was able to produce latency estimations with less than 20% error in nearly 80% of all cases.

4.1.2. Resource requirement

In this paper, the server resource requirement of each client is measured as the bandwidth requirement. Following the work in [31], the bandwidth requirement of each client on its target server $R^{\mathrm{T}}_{c_i}$ in a server-based architecture can be estimated in advance as follows.

First we assume that the size of the message each client c_i sends to its target server is L bytes, the client's sending frequency, i.e., frame rate, is T, and the number of concurrent clients in c_i's zone is N. On receiving a message from client c_i, the target ser-

[4] This server can be the one that the clients first connect to before joining the virtual world. It is regarded as the master server, which monitors the status and locations of other servers that are hosting the zones, and manages users' related information such as username, password, etc.

[5] This machine is also the server that executes our assignment algorithms.

ver replies c_i with the state updates of *all* the clients in c_i's zone, which requires LN bytes.[6] Hence, the bandwidth requirement of a client c_i on its target server $R^{T}_{c_i}$ can be calculated as

$$R^{T}_{c_i} = (L + LN)T = (N + 1)LT. \quad (14)$$

Based on the definition in the previous section, the bandwidth requirement of a client c_i on its contact server can be calculated as

$$R^{C}_{c_i} = \begin{cases} 2(N+1)LT, & \text{if } s^c(c_i) \neq s^t(c_i), \\ 0, & \text{if } s^c(c_i) = s^t(c_i). \end{cases} \quad (15)$$

The total bandwidth requirement on a server s_j can then be easily calculated from the number of clients that are interacting on it (i.e., it is the target server for these clients), and the number of clients that are connecting to it (i.e., it is the contact server for these clients).

4.2. Dynamic executions of assignment algorithms

Another important consideration is the dynamic property of DVEs. During the course of interactions in the virtual world, clients may move from one zone to another, new clients may join, existing clients may also leave the virtual world. An obtained client assignment may not be good after some time. Thus, the proposed two-phase algorithm needs to be executed again to ensure good client assignments.

Our concern here is how frequently the client assignment algorithms should be executed. In a recent study [32] of the very popular MMOG Lineage II developed by NCsoft, Korea, the authors found that the average value of session durations, i.e., the average time a player keeps on playing the game, is about 3 h, which is rather long. This may indicate that we do not have to execute the assignment algorithms frequently. Actually, in a typical deployment of a large-scale DVE, the master server that executes the assignment algorithms would constantly monitor the system performance, i.e., the number of client that are with QoS. Only when this value become lower than a pre-defined threshold, then re-assignment should take place.

In addition, we should note that dynamic execution of the client assignment algorithms may incur inter-server bandwidth consumption due to the migrations of zones or clients across servers. Nevertheless, we believe this cost is much smaller compared to the bandwidth consumption due to the interactions of users in the DVE. This suggests that the bandwidth cost of re-executing the assignment algorithms is not a big concern, given that re-assignment does not occur frequently. Indeed, a similar assumption is used in the context of the Web replica placement problem [22].

Another situation that reassignment needs to be done is when a server or a subset of server is suddenly down, because of hardware failure, for example. This reassignment process is expected to be completed relatively fast. First, the reassignment in this case only involves those clients that are currently connected to the failed servers, i.e., the reassignment is not a global assignment involving all the clients in the system. Second, the master server of the system must constantly monitor the DVE servers, and any server failure should be detected quickly. Moreover, our greedy assignment algorithms are very fast, for example, as shown in Section 6, it takes less than 1 s only to complete the assignment of 5000 clients grouped in 400 zones to 20 geographically distributed servers.

5. Evaluation methodology

5.1. Network models

In our simulations, we used both synthetic topologies generated by the popular topology generator BRITE [33] and real Internet topologies. Table 1 lists the topologies used in this paper. We use BRITE to generate both flat and hierarchical topologies based on the well-known Waxman and Barabasi–Albert model (F–W, F–BA, H–BA/W). The Waxman model [34] considers all pairs of nodes, and then decides whether to add a link between any two nodes with a probability that depends on the distance between these two nodes and the longest distance between any two nodes in the network. The Barabasi–Albert model [35] generates network topologies that exhibit power-laws as observed in the seminal paper by Faloutsos et al. [36] in 1999.

To complement the synthetic topologies generated by BRITE, we use a real, flat Internet topology collected from NLANR [37]. For diversity, we also collect a real AS-level topology (generated by processing the Border Gateway Protocol (BGP) routing tables) with 110 nodes from [38], and use the DFN (German Research Network) topology [39] with 30 nodes as the router-level topology to

[6] For simplicity, we assume that all messages have the same size. This assumption is also used in [31].

Table 1
Network topologies

Topology	Nodes	Links
Flat, Waxman (F–W)	3000	6000
Flat, Barabasi-Albert (F–BA)	3000	5997
Flat, real (F–R)	3024	5192
Hierarchical, Barabasi-Albert/Waxman (H–BA/W)	3000	6197
Hierarchical, real/real (H–R/R)	3300	13,442

construct a realistic hierarchical topology (H–R/R). While these topologies are not complete, they at least partially reflect the "true" topology of the Internet, which may have great impacts on the simulation results.

For simplicity, we assume that the round-trip network delay between any two nodes in the network topology is proportional to the number of link-hops between them. This assumption is similar to the one used in most of the previous work [22,40]. In fact, a recent Internet measurement [41] also showed that round-trip delay is well-correlated with network hop counts. Moreover, to obtain more realistic simulations results, in calculating the network delays, we use both shortest-path routing and AS-level hierarchical routing [42] where possible. More specifically, when the network topology is a hierarchical one, AS-level hierarchical routing can be employed. However, when flat topologies are used, shortest-path routing is the only choice.

The AS-level hierarchical routing is a more realistic routing strategy for our simulations, since to some extent it may reflect the "true" routing practice in the current Internet [42]. With this strategy, first we calculate the AS-level shortest path. Then, to get the distance between any two nodes at the router-level, the AS-level shortest path is followed, and for each AS on that shortest path, the router-level shortest path (within that AS) is used.

5.2. Workload models

5.2.1. Client distributions

To fully evaluate the performance of the proposed assignment algorithms, we simulate several different client distributions in both the virtual world and the physical world (the network). The number of clients may be larger in some specific zones of the virtual world than others, due to the clustering of clients in some "hot" zones. For example, in online games, clients may be clustered in the zones with large amounts of game resources such as energy, gold, etc. In the physical world, due to the differences in time zones of geographically distributed clients, at a specific time, the number of online clients in the DVE may be quite different for different geographic regions [43].

We simulate the clustering behavior of clients in the virtual world by randomly selecting some zones to have more clients than other zones. To simulate the clustering of clients in the physical world, some nodes in the network topology are randomly selected to have a larger number of clients than the rest nodes. We have simulated a large number of different scenarios by changing the number of clusters and number of clients in each clusters, and obtained similar results. Hence, in this paper, we use the following method to generate different client distributions. For the clustered distribution in the virtual world, the number of clients in a clustered zone is 10 times larger than that in a non-clustered zone. The clustered distribution for the physical world is generated in a similar manner. Table 2 shows the combination of different virtual world (VW) and physical world (PW) distributions.

5.2.2. Physical world-virtual world correlation

In general, we should note that clients gathering in the same zone of a DVE may not necessarily close to each other in terms of their physical locations. On the other hand, it is natural to observe that clients that are close to each other in their physical locations (e.g., from the same country or the same geographic region) tend to gather in a specific zone of the virtual world due to their common cultural preferences. These phenomena may have great impacts on the performance of the proposed algorithms.

To model the correlation between clients' locations in the physical world and those in the virtual world, we use a correlation parameter δ, where $0 \leqslant \delta \leqslant 1$ [8]. The higher the value of δ is, the stronger the tendency for clients from the close geo-

Table 2
Distribution types

Type	Clusters in PW	Cluters in VW
0	No	No
1	Yes	No
2	No	Yes
3	Yes	Yes

graphic locations to gather in specific zones of the virtual world.

5.2.3. DVE configurations

Different DVE configurations are used for performance evaluation. A specific DVE configuration is determined by the number of servers, the number of zones, the number of clients and the total resource capacity of the system. We use the notation *number of servers–number of zones–number of clients-capacity* to denote a DVE configuration. For example, the notation 20s–400z–5000c–2500cp means that the DVE has 20 servers, 400 zones, 5000 clients and 2500 Mpbs server bandwidth in total. In investigating the proposed client assignment algorithms, we use two DVE configurations, 5s–30z–400c–200cp (small) and 20s–400z–5000c–2500cp (large). The small configuration is only used to enable the comparison between our algorithms and the optimal solution produced by the MILP solver lp_solve [28]. In all other cases, we use the large configuration.

5.3. Performance measures

Two main performance measures, namely the percentage of clients with QoS in the system (measuring the interactivity of the system), denoted as *pQoS*, and the server resource utilization, i.e., bandwidth consumption, (measuring the cost of the algorithms), are of interest in the analysis. Results presented in this paper are obtained by averaging the results of 50 simulation runs.

5.4. Default settings

Unless otherwise stated, the following assumptions and default values are used in the simulations. The clients are uniformly distributed in the physical world as well as in the virtual world. For estimating bandwidth requirement as in [31], the input sending frequency of each client (frame rate) is set to 25 messages per second, and the size of each input or update is 100 bytes, which are close to real settings [44]. The maximum round-trip delay between any two nodes in the network topology is set to 300 ms. The interactivity requirement, i.e., the DVE delay bound D is set to 150 ms. The default DVE configuration is 20s–400z–5000c–2500cp.

6. Results

6.1. Normal inter-server networks

In this section, we study the performance of client assignment algorithms in the case that the networks between servers are not well-provisioned, thus the word "normal". Hence, the client assignment algorithms considered in this section include RanZ/VirC, GreZ-1/VirC and GreZ-2/VirC. Note that these algorithms do not incur inter-server communications.

Figs. 5 and 6 show the CDF of client–server round-trip communication delays for our algorithms in the small and large configuration with different network topologies. In all cases, we observe

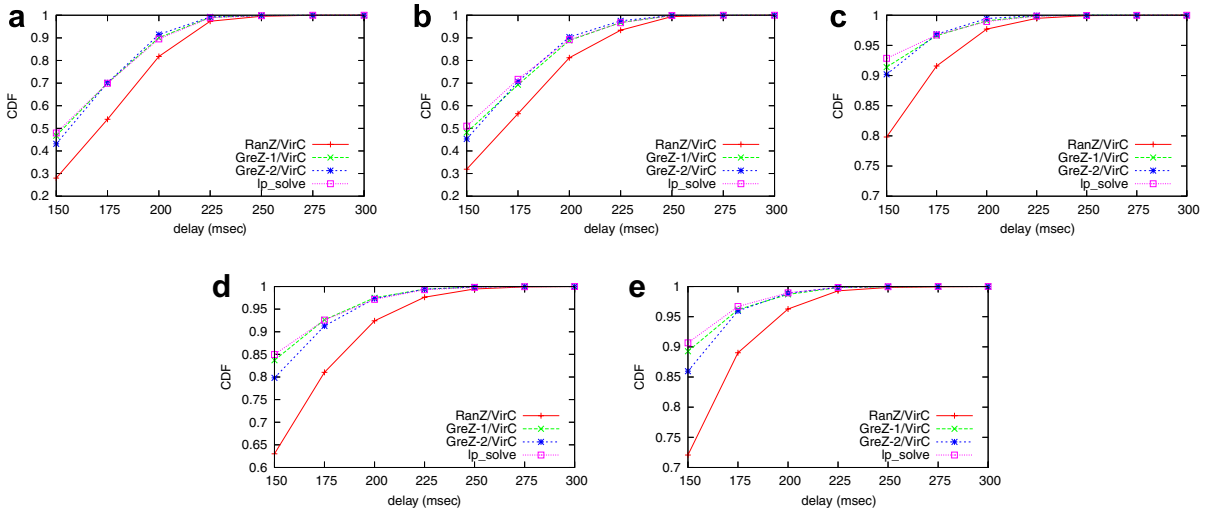

Fig. 5. Impacts of client assignment – small configuration, normal inter-server networks. (a) F–W topology; (b) F–BA topology; (c) F–R topology; (d) H–BA/W topology; (e) H–R/R topology.

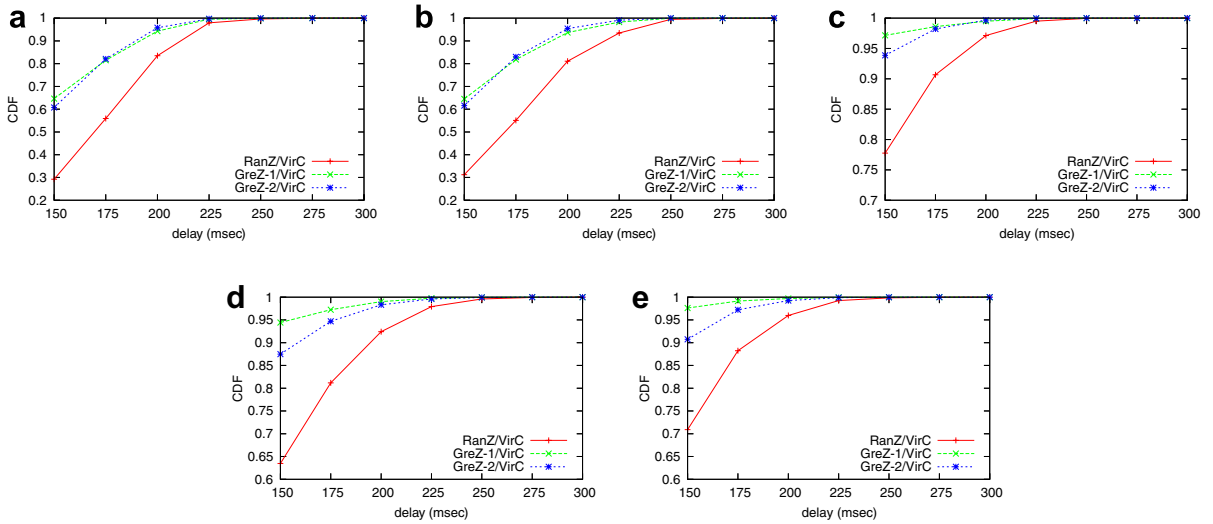

Fig. 6. Impacts of client assignment – large configuration, normal inter-server networks. (a) F–W topology; (b) F–BA topology; (c) F–R topology; (d) H–BA/W topology; (e) H–R/R topology.

that the GreZ-1/VirC and GreZ-2/VirC significantly outperforms the delay-oblivious RanZ/VirC algorithm in terms of *pQoS*. We also observed that among the three algorithms (RanZ/VirC, GreZ-1/VirC and GreZ-2/VirC), GreZ-1/VirC has the best performance in terms of *pQoS*. This can be explained by the fact that GreZ-1/VirC aims to directly minimize the total number of clients without QoS in the system, while GreZ-2/VirC tries to assign a zone to a server that is close (in terms of network delay) to the center of mass of the client population in that zone. The performance improvements of GreZ-1/VirC over RanZ/VirC are 63% (F–W), 58% (F–BA), 16% (F–R), 35% (H–BA/W) and 25% (H–R/R) for small configuration, and 116% (F–W), 110% (F–BA), 26% (F–R), 48% (H–BA/W) and 38% (H–R/R) for large configuration.

We also note that the *pQoS* values of GreZ-1/VirC are close to the optimal results given by the branch-and-bound algorithm implemented in the lp_solve software. Note that lp_solve can only be applied to small size DVEs, i.e., the small configuration. The average execution time of lp_solve for this configuration is around 1 s. For the large configuration, the results by lp_solve are not shown since the execution time was too long (not finished after more than 10 h), which is clearly impractical for interactive application like DVEs which need timely assignment decisions. In both small and large configurations, all of our proposed algorithms took less than 0.5 s to execute.

6.1.1. Impacts of correlations

The performance of our proposed algorithms with different physical world-virtual world correlation values is shown in Fig. 7. It is observed that the *pQoS* values of both GreZ-1/VirC and GreZ-2/VirC increase significantly with the correlation value, while the result of the delay-oblivious algorithm RanZ/VirC does not change much. This demonstrates the effectiveness of the greedy algorithms when the clients in each zone of the virtual world

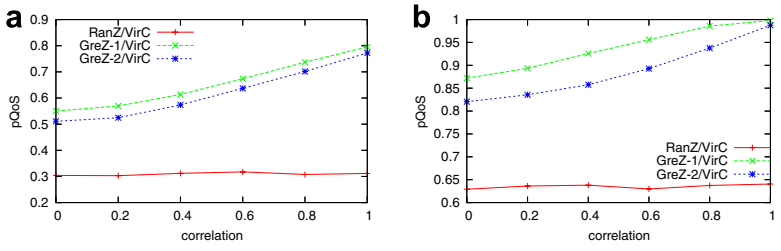

Fig. 7. Impacts of physical – virtual world correlation, normal inter-server networks. (a) F–BA and (b) H–BA/W.

are close to each other in the physical world. In all cases, GreZ-1/VirC is still the best algorithm in terms of *pQoS*. However, it is also noted that GreZ-2/VirC is a good alternative when the correlation value is very high.

6.1.2. Impacts of client distributions

The simulation results for all assignment algorithms with different client distributions are shown in Fig. 8. In all cases, both GreZ-1/VirC and GreZ-2/VirC are always better than RanZ/VirC, regardless of the client distributions, while GreZ-1/VirC is the best algorithm.

The impacts of client distributions on *pQoS* of the greedy assignment algorithms are clear: the dominant factor seems to be the client distribution in the virtual world, for example, the *pQoS* is better when the distribution type is 0 and 1, i.e., clients are distributed uniformly in the virtual world. This observation can be explained if we look at Fig. 8(b), which shows the resource utilization for each distribution type. The distribution type 0 and type 1 use much less resource than type 2 and type 3, thus GreZ-1/VirC and GreZ-2/VirC perform better if clients are not highly clustered in the virtual world.

6.1.3. Impacts of imperfect inputs

The simulation results obtained above for the client assignment algorithms are based on the assumption that we have perfect information about the network delays between clients and servers. In practice, we usually have rough estimations of network delays rather than perfectly accurate information. To simulate the estimation error, we use the method presented in [22]. More specifically, we apply an error factor e to the perfect input data, i.e., assuming the exact value of the delay is d, then the delay value input to our algorithms is uniformly distributed in the range $[\frac{d}{e}, de]$. Note that although the algorithms use inaccurate inputs to make assignment decisions, their resulted performances, i.e., *pQoS*, are still calculated based on the actual network delays after the assignment is done. As in [22], we use three different values of e: 1.2, 2 and 4, representing the inaccuracies of the popular network delay estimation tools King [26] and IDMaps [29].

Fig. 9 shows a typical simulation result obtained with the inaccurate estimations of network delays. Despite the imperfect knowledge in network delay, the greedy algorithms are still much better than RanZ/VirC which does not consider network delays. In particular, for $e = 1.2$, GreZ-1/VirC is still the best algorithm in terms of interactivity, and its *pQoS* only decreases slightly compared to the cases that use perfect information. However, when the estimation error becomes large, i.e., $e = 2$ or 4, we see that GreZ-2/VirC is the best algorithm. Fig. 9 shows that GreZ-2/VirC exhibits low

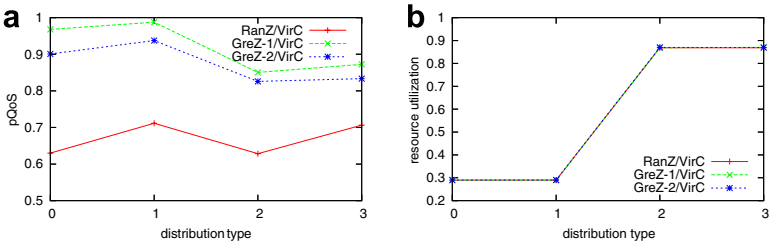

Fig. 8. Impacts of client distributions on client assignment algorithms, normal inter-server networks, H–BA/W. (a) *pQoS*. (b) Resource utilization.

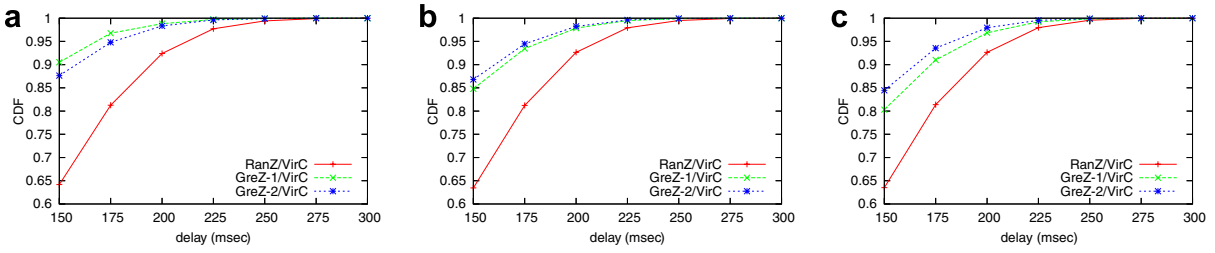

Fig. 9. Impacts of imperfect inputs, normal inter-server networks, H–BA/W. Error factors are 1.2 (a), 2 (b) and 4 (c).

sensitivity to input data estimation errors, even if the error is very large, i.e., $e = 4$. This suggests that GreZ-2/VirC is a more suitable choice for the client assignment problem under Internet environments, where perfectly accurate input data for assignment algorithms is usually impossible or very costly to obtain.

6.2. Well-provisioned inter-server networks

In this section, we study the performance of client assignment algorithms in the case that the networks between servers are well-provisioned. The client assignment algorithms considered in this case include RanZ/CloC, GreZ-1/CloC, GreZ-2/CloC, RanZ/GreC, GreZ-1/GreC and GreZ-2/GreC. To simulate the well-provisioned inter-server networks, we reduce the network latency between any two geographically distributed servers to 50% of the actual latency values obtained from the topology generator, as in [21]. Note that we do not show the impacts of client distributions and correlations in this case, since they are similar to the case of normal inter-server work.

Tables 3–6 show the *pQoS* and resource utilization, while Fig. 10(a) and (b) shows the CDFs of communication delays for the proposed client assignment algorithms for both small and large configurations with different network topologies. The first observation is that all the algorithms that have CloC as the refined assignment perform badly, more specifically, even worse (in terms of both *pQoS* and resource utilization) than the algorithms that have VirC as the refined assignment,[7] although CloC's principle is widely used (and produces good results) in the context of Web replica placement problem. Note that, for comparison, the performances of algorithms with the refined assignment VirC are also included in Tables 3–6 (the values inside the brackets of the first three rows of each table).

We believe that the main reason for the ineffectiveness of CloC when applying to DVEs is that the network path from a client to its target server via that client's closest server may not be the shortest paths in many cases, despite the inter-server networks are well-provisioned. Hence, selecting the closest servers as contact servers for clients may result in high communication delays, and incur much more resource consumption. This clearly illustrates the difference between the client assignment problem in DVEs and in Web environments: the "target server" of a client in a DVE may not be its closest server, but this is true for the latter.

We have also tested CloC with higher delay reduction factors such as 70% or 90% as suggested in [21]. However, the final conclusion is that in most cases, CloC *worsens* the *pQoS* obtained from the initial assignment, and significantly increases the resource utilization. Hence, we deem it inappropriate to solve the client assignment problem in DVEs, and decide not to further investigate it in this paper. From this point, for well-provisioned inter-server networks, we only consider the following greedy algorithms: RanZ/GreC, GreZ-1/GreC and GreZ-2/GreC.

6.2.1. Impacts of inter-server network delays

Fig. 11 show the impacts of the delay reduction factor on our assignment algorithms. Note that when the reduction factor is 0, i.e., the inter-server networks are not well-provisioned, GreC is effectively equal to VirC. One important observation here is, unlike RanZ/GreC, algorithm GreZ-1/GreC and GreZ-2/GreC do not heavily depend on the reduction factor. This result suggests that GreZ-1/GreC and GreZ-2/GreC are still able to perform well even in the case there is little delay reduction in the inter-server networks.[8] This is because the initial greedy assignment phase already provides a good value of *pQoS*, and there may be not much room for further improvement in the refined assignment. Nevertheless, whenever low-delay inter-server networks are available, the refined assignment phase GreC is still beneficial.

6.2.2. Impacts of imperfect inputs

We evaluate the performances of RanZ/GreC, GreZ-1/GreC and GreZ-2/GreC with imperfect input data. Similar to the previous experiment, we use three different values of the error factor *e*: 1.2, 2 and 4, to add random noise to the network delays.

[7] Algorithms with VirC as the refined assignment do not require well-provisioned inter-server networks.

[8] We should note that providing low-delay networks that interconnect geographically distributed servers may be expensive, especially in the current Internet where well-known network QoS techniques like IntServ or DiffServ have not been widely deployed. Nevertheless, without a dedicated, private high-speed network, we still can reduce the communication delays between distributed servers, by closely monitoring the routing paths among these servers [15].

Table 3
Impacts of client assignments on *pQoS*, well-provisioned inter-server networks, small configuration

Algo. vs. Topo.	F–W	F–BA	F–R	H–BA/W	H–R/R
RanZ/CloC (RanZ/VirC)	0.24 (0.28)	0.26 (0.32)	0.66 (0.8)	0.58 (0.63)	0.65 (0.72)
GreZ-1/CloC (GreZ-1/VirC)	0.39 (0.47)	0.43 (0.48)	0.8 (0.91)	0.71 (0.83)	0.75 (0.89)
GreZ-2/CloC (GreZ-2/VirC)	0.4 (0.43)	0.39 (0.45)	0.77 (0.9)	0.73 (0.8)	0.76 (0.86)
RanZ/GreC	0.28	0.33	0.81	0.71	0.79
GreZ-1/GreC	0.48	0.49	0.93	0.85	0.91
GreZ-2/GreC	0.44	0.46	0.91	0.82	0.87
lp_solve	0.49	0.51	0.94	0.86	0.92

Table 4
Impacts of client assignments on resource utilization, well-provisioned inter-server networks, small configuration

Algo. vs. Topo.	F–W	F–BA	F–R	H–BA/W	H–R/R
RanZ/CloC (RanZ/VirC)	0.99 (0.61)	0.99 (0.61)	0.95 (0.61)	0.99 (0.61)	0.99 (0.61)
GreZ-1/CloC (GreZ-1/VirC)	0.97 (0.61)	0.95 (0.61)	0.89 (0.61)	0.97 (0.61)	0.97 (0.61)
GreZ-2/CloC (GreZ-2/VirC)	0.97 (0.61)	0.95 (0.61)	0.9 (0.61)	0.95 (0.61)	0.96 (0.61)
RanZ/GreC	0.7	0.71	0.68	0.8	0.78
GreZ-1/GreC	0.64	0.64	0.62	0.66	0.65
GreZ-2/GreC	0.63	0.63	0.62	0.67	0.66
lp_solve	0.95	0.92	0.65	0.71	0.67

Table 5
Impacts of client assignments on *pQoS*, well-provisioned inter-server networks, large configuration

Algo. vs. Topo.	F–W	F–BA	F–R	H–BA/W	H–R/R
RanZ/CloC (RanZ/VirC)	0.24 (0.29)	0.25 (0.31)	0.67 (0.77)	0.63 (0.64)	0.7 (0.71)
GreZ-1/CloC (GreZ-1/VirC)	0.50 (0.65)	0.51 (0.64)	0.76 (0.96)	0.79 (0.94)	0.81 (0.97)
GreZ-2/CloC (GreZ-2/VirC)	0.51 (0.61)	0.51 (0.61)	0.78 (0.94)	0.80 (0.88)	0.82 (0.91)
RanZ/GreC	0.34	0.35	0.82	0.81	0.89
GreZ-1/GreC	0.66	0.65	0.97	0.97	0.99
GreZ-2/GreC	0.63	0.62	0.95	0.93	0.96

Table 6
Impacts of client assignments on resource utilization, well-provisioned inter-server networks, large configuration

Algo. vs. Topo.	F–W	F–BA	F–R	H–BA/W	H–R/R
RanZ/CloC (RanZ/VirC)	0.99 (0.58)	0.99 (0.58)	0.99 (0.58)	0.98 (0.58)	0.99 (0.58)
GreZ-1/CloC (GreZ-1/VirC)	0.99 (0.58)	0.93 (0.58)	0.93 (0.58)	0.99 (0.58)	0.98 (0.58)
GreZ-2/CloC (GreZ-2/VirC)	0.99 (0.58)	0.9 (0.58)	0.9 (0.58)	0.97 (0.58)	0.99 (0.58)
RanZ/GreC	0.84	0.74	0.74	0.91	0.88
GreZ-1/GreC	0.66	0.59	0.58	0.62	0.6
GreZ-2/GreC	0.64	0.59	0.59	0.67	0.66

The simulation results are shown in Fig. 12. Note that we also include the results for RanZ/VirC, GreZ-1/VirC and GreZ-2/VirC (the algorithms proposed for normal inter-server networks) for comparison.

From the simulation results, we see that when the delay estimation error is small, i.e., $e = 1.2$, all the algorithms that use the knowledge of network delays are still better than the delay-oblivious algorithm RanZ/VirC in terms of *pQoS*. However, when the error becomes larger, i.e., $e = 2$, RanZ/GreC performs even worse than RanZ/VirC in terms of both *pQoS* and resource utilization. In this case, although GreZ-1/GreC and GreZ-2/GreC are still better than RanZ/VirC, they are not as good as GreZ-1/VirC and GreZ-2/VirC. Moreover, when e is very large, i.e., $e = 4$, GreZ-1/GreC and GreZ-2/GreC perform comparably to RanZ/VirC, which is much worse

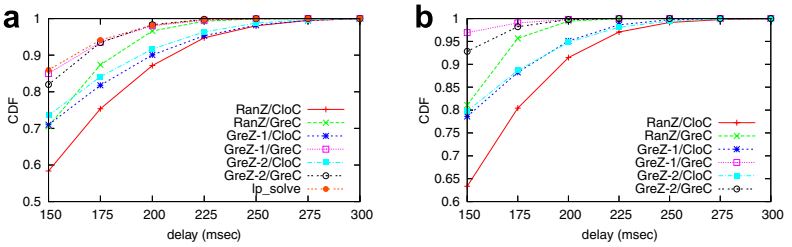

Fig. 10. Impacts of client assignment algorithms, well-provisioned inter-server networks, H–BA/W. (a) Small configuration. (b) Large configuration.

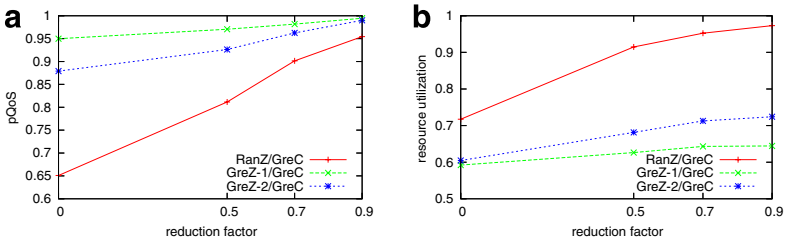

Fig. 11. Impacts of inter-server network delays, H–BA/W. (a) *pQoS*. (b) Resource utilization.

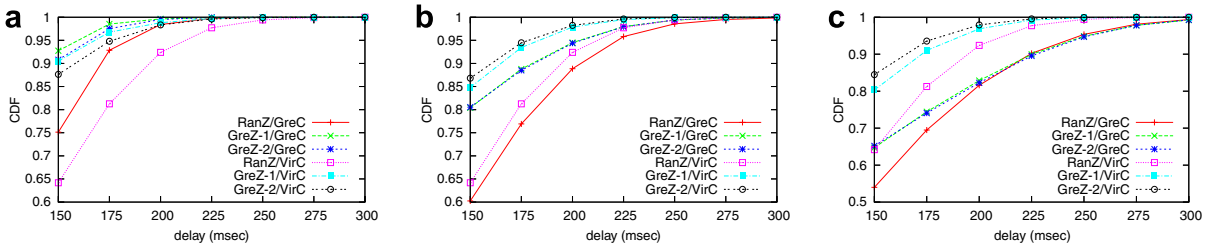

Fig. 12. Impacts of imperfect inputs, well-provisioned inter-server networks, H–BA/W. Error factors are 1.2 (a), 2 (b) and 4 (c).

than GreZ-1/VirC and GreZ-2/VirC. This important observation suggests that the refined assignment algorithm GreC is quite sensitive to the estimation error. Hence, GreC should not be used when the network delays input to the assignment algorithms are highly inaccurate. In such scenario, GreZ-2/VirC is the best choice, since it not only exhibits low-sensitivity to delay estimation errors, but also does not require well-provisioned inter-server networks and thus incurs less resource utilization.

7. Conclusions

Supporting large-scale DVEs with thousands of simultaneous clients interacting in real-time is a challenging task. In this paper, we have described the geographically distributed server architecture for DVEs. In this architecture, multiple geographically distributed servers are connected with each other, usually via well-provisioned networks to provide low-latency inter-server communications, and the large virtual world is partitioned into distinct zones to distribute load among the servers. The client assignment problem arises when assigning participating clients to servers to enhance the interactivity of the DVE. In this paper, we have developed new formulations and algorithms to deal with this problem.

The main contributions of this paper are as follows:

- We have proposed and formulated a new problem, termed the *client assignment* problem. This problem essentially concerns how to assign clients to distributed servers to reduce the client–server communication delays.

- We have proposed a two-phase approach, which consists of an initial assignment and a refined assignment, to solve the client assignment problem. Each phase is proved to be NP-hard. Therefore, we have developed several simple yet effective two-phase algorithms to address the client assignment problem.
- To evaluate the proposed algorithms, we have developed a realistic simulation model, in which a wide range of network topologies are used. We have also implemented real-world routing protocols, and modelled network delays according to real Internet measurements. Different client distributions and physical world–virtual world correlations have been used to simulate different scenarios in real applications.
- We have conducted extensive simulations to evaluate the effectiveness of the proposed client assignment algorithms. We find that all greedy algorithms, especially GreZ-1/VirC, GreZ-1/GreC, GreZ-2/VirC and GreZ-2/GreC, that utilize the knowledge of network delays outperform the delay-oblivious algorithm, even though the delays input to these algorithms may be inaccurate. We also find that, when the delay estimation error becomes very large, the GreZ-2/VirC algorithm performs the best. Moreover, GreZ-2/VirC does not require well-provisioned inter-server networks. This property is of great significance for practical deployments, since usually an Internet-wide well-provisioned network is quite expensive.

In summary, we believe that our findings in this paper, especially the two-phase approach with the greedy assignment algorithms will be very useful to the designers and researchers of large-scale DVEs. For the future work, we would like to investigate the effectiveness of our proposed algorithms on a real, geographically distributed network testbed like PlanetLab [45]. Another interesting direction is to combine processing delay and network delay in our client assignment problem.

Acknowledgement

The authors would like to thank the reviewers of the first version of this paper for their constructive comments and valuable suggestions which help to improve the quality and presentation of this paper.

References

[1] S. Singhal, M. Zyda, Networked Virtual Environments: Design and Implementation, Addison-Wesley, Reading, MA, 1999.
[2] S. Zhou, W. Cai, B.S. Lee, S.J. Turner, Time–space consistency in large-scale distributed virtual environments, ACM Transactions on Modeling and Computer Simulation 14 (1) (2004) 31–47.
[3] Everquest, http://www.everquest.com.
[4] Ultima Online, http://www.uo.com.
[5] Multihoming, http://en.wikipedia.org/wiki/Multi-homed.
[6] M. Mauve, S. Fischer, J. Widmer, A generic proxy system for networked computer games, in: Proc. of NetGames, 2002.
[7] D. Bauer, S. Rooney, P. Scotton, Network infrastructure for massively distributed games, in: Proc. of NetGames, 2002.
[8] C.D. Nguyen, F. Safaei, P. Boustead, Comparison of distributed server architectures in providing immersive audio communication to massively multi-player online games, in: Proc. of ATNAC, 2004.
[9] F. Safaei, P. Boustead, C.D. Nguyen, J. Brun, M. Dowlatshahi, Latency driven distribution: infrastructure needs for participatory entertainment applications, IEEE Communication Magazine on Entertainment Everywhere: System and Networking Issues in Emerging Network-Centric Entertainment Systems 43 (5) (2005).
[10] B.P.J. Mulligan, Developing Online Games: An Insider's Guide, New Riders Games, 2003.
[11] Microsoft's Age of Empires, http://www.microsoft.com/games/empires/.
[12] J. Smed, T. Kaukoranta, H. Hakonen, Aspects of networking in multiplayer computer games, in: Proc. of the International Conference on Application and Development of Computer Games in the 21st Century, 2001, pp. 74–81.
[13] T. Henderson, S. Bhatti, Networked games: a QoS-sensitive application for QoS-insensitive users? in: Proc. of the ACM SIGCOMM, 2003.
[14] L. Pantel, L. Wolf, On the impact of delay on real-time multiplayer games, in: Proc. of NOSSDAV, 2002, pp. 23–29.
[15] D.G. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, Resilient overlay networks, in: Proc. of the 18th ACM SOSP, 2001.
[16] Detour project, http://www.cs.washington.edu/research/networking/detour/.
[17] D.S. Hochbaum, Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, Boston, MA, 1997.
[18] D.N.B. Ta, S. Zhou, A dynamic load sharing algorithm for massively multi-player online games, in: Proc. of the 11th IEEE International Conference on Networks, 2003.
[19] J. Lui, M. Chan, An efficient partitioning algorithm for distributed virtual environment systems, IEEE Transaction on Parallel and Distributed Systems 13 (3) (2002).
[20] J. Lui, Constructing communication sub-graphs and deriving and optimal synchronization interval for distributed virtual environment systems, IEEE Transactions on Knowledge and Data Engineering 13 (5) (2001) 778–792.
[21] K.W. Lee, B.J. Ko, S. Calo, Adaptive server selection for large scale interactive online games, Computer Networks 49 (2005) 84–102.
[22] L. Qiu, V. Padmanabhan, G. Voelker, On the placement of web server replicas, in: Proc. of IEEE INFOCOM, 2001.

[23] E. Cronin, S. Jamin, C. Jin, A.R. Kurc, D. Raz, Y. Shavitt, Constrained mirror placement on the internet, IEEE Journal on Selected Areas in Communications (2002).
[24] H. Romeijn, D.R. Morales, A class of greedy algorithms for the generalized assignment problem, Discreet Applied Mathematics 103 (2000) 209–235.
[25] H. Feltl, R. Raidl, An improved hybrid genetic algorithm for the generalized assignment problem, in: Proc. of ACM SAC, 2004.
[26] K. Gummadi, S. Saroiu, S. Gribble, King: Estimating latency between arbitrary internet end hosts, in: Proc. of the ACM SIGCOMM IMW, 2002.
[27] Branch and bound method, http://www.nist.gov/dads/HTML/branchNbound.html.
[28] Mixed Integer Linear Programming solver lp_solve, http://groups.yahoo.com/group/lp_solve.
[29] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, IDMaps: A global internet host distance estimation service, IEEE/ACM Transactions on Networking 9 (5) (2001).
[30] K. Moore, J. Cox, S. Green, SONAR: A Network Proximity Service (Internet Draft), http://www.netlib.org/utk/projects/sonar, 1996.
[31] J. Pellegrino, C. Dovrolis, Bandwidth requirement and state consistency in three multiplayer game architectures, in: Proc. of the ACM NetGames, 2003.
[32] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, E. Yuk, Traffic characteristics of a massively multi-player online role playing game, in: Proc. of NetGames, 2005.
[33] BRITE Internet Topology Generator, http://www.cs.bu.edu/brite.
[34] B.M. Waxman, Routing of multipoint connections, IEEE Journal on Selected Areas in Communications 9 (1988) 1617–1622.
[35] A. Barabasi, R. Albert, Emergence of scaling in random networks, Science (1999) 509–512.
[36] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the Internet topology, in: ACM SIGCOMM, 1999.
[37] The NLANR project, http://moat.nlanr.net/Routing/rawdata.
[38] SSF: Gallery of Network Models, http://www.ssfnet.org/Exchange/gallery/.
[39] O. Heckmann, M. Piringer, J. Schmitt, R. Steinmetz, Generating realistic ISP-level network topologies, IEEE Communication Letters, 2003.
[40] R.G.P. Radoslavov, D. Estrin, Topology-informed internet replica placement, in: Proc. of Web Caching and Content Distribution Workshop, 2001.
[41] K. Obraczka, F. Silva, Network latency metrics for server proximity, in: Proc. of the IEEE Globecom 2000, 2000.
[42] H. Tangmunarunkit, R. Govindan, S. Shenker, D. Estrin, The impact of routing policy on Internet paths, in: Proc. of IEEE INFOCOM, 2001.
[43] W.C. Feng, W.C. Feng, On the geographic distribution of online game servers and players, in: Proc. of NetGames, 2003.
[44] A. Abdelkhalek, A. Bilas, A. Moshovos, Behavior and performance of interactive multi-player game servers, Special Issue of Cluster Computing: the Journal of Networks, Software Tools and Applications, 2002.
[45] PlanetLab, http://www.planet-lab.org.

Duong Nguyen Binh Ta is currently a Research Engineer in the Advanced Computing Programe, Institute of High Performance Computing, Singapore. He received his Ph.D. in Computer Science from the School of Computer Engineering, Nanyang Technological University, Singapore, and B.Eng. in Computer Engineering from the Department of Information Technology, University of Technology, Ho Chi Minh city, Vietnam. His current research interests include distributed virtual environments, Internetworking, grid computing and optimization.

Suiping Zhou is currently an Assistant Professor in the School of Computer Engineering at Nanyang Technological University, Singapore. Previously, he was a Post-doctoral fellow at Weizmann Institute of Science, Israel. He received his B.Eng., M.Eng. and Ph.D. in Electrical Engineering from Beijing University of Aeronautics and Astronautics, PR China. His current research interests include distributed interactive applications, hybrid control systems and human behavior representation for virtual training systems.