Client assignment problems for distributed interactive computing

Zhang, Lu

2013

Zhang, L. (2013). Client assignment problems for distributed interactive computing.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/51873>

<https://doi.org/10.32657/10356/51873>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**CLIENT ASSIGNMENT PROBLEMS FOR
DISTRIBUTED INTERACTIVE COMPUTING**

**ZHANG LU
SCHOOL OF COMPUTER ENGINEERING
2013**

CLIENT ASSIGNMENT PROBLEMS FOR DISTRIBUTED INTERACTIVE COMPUTING

ZHANG LU

Supervisor: Dr. Tang Xueyan

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2013

Abstract

The steady growth in broadband Internet accesses and PC penetration is leading to a shift in emphasis from individual computers to distributed interactive computing that allows multiple participants at different locations to interact with each other in real time. Recent years have witnessed rapid development of Distributed Interactive Applications (DIAs) in many areas such as interactive digital media and entertainment, distributed interactive simulation, and collaborative computer-aided design and engineering. Interactivity is a primary performance measure of DIAs owing to their human-in-the-loop nature. To support graceful interactions, it is required that when a participant of the DIA performs an operation, other participants are able to see the effect of the operation quickly. Network latency is a major barrier to propagating and reflecting the effects of operations timely. Wide geographical spreads of participants in large-scale DIAs necessitate distributed deployment of servers to combat network latency and achieve high interactivity. In a distributed server architecture, the interactivity performance depends not only on client-to-server latencies but also inter-server latencies. Meanwhile, the consistency and fairness requirements of DIAs, which are important to providing pleasant user experiences, may introduce artificial synchronization delays in the interaction that can also influence the interactivity performance. All these factors are directly affected by how the clients are assigned to the servers.

In this thesis, we investigate how to assign clients to appropriate servers in an effective and efficient manner for enhancing the interactivity performance of DIAs. Based on whether the application's state changes due to the passing of time, DIAs can be classified into discrete

DIAs and continuous DIAs. We study the client assignment problems for both types of DIAs. We first mathematically model the interactivity performance of these two types of DIAs, taking the consistency and fairness requirements into consideration. Based on the analysis, we formulate the client assignment problems of discrete and continuous DIAs as combinatorial optimization problems. We show that both client assignment problems are NP-complete via polynomial time reductions from existing NP-complete problems. Then, a number of heuristics, including both centralized and distributed algorithms, are proposed for fast computation of good client assignments to each problem. The approximation ratios of these algorithms are theoretically analyzed and examples are given to show the tightness of these ratios. The performance of these algorithms is also experimentally evaluated using real Internet latency data. The results show that our proposed greedy and modification-based algorithms normally perform close to the optimal assignments, and they significantly outperform the intuitive *Nearest-Server Assignment* algorithm that assigns each client to its nearest server. For the client assignment problem of discrete DIAs, we further show that there exists a polynomial-time optimal solution for the special case of tree network topologies. An efficient algorithm is developed to compute the optimal client assignment.

Acknowledgments

I would like to express my sincere gratitude to Dr. Tang, Xueyan, for his guidance, support and patience. My thanks also go to all my friends in PDCC for their help. Last, but not least, I would like to thank my wife Yang for her support and encouragement during the past four years. My parents, Xiao and Yun, receive my deepest gratitude and love for their inspiration and the many years of support.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vii
List of Tables	ix
List of Notations	x
 1 Introduction	 1
1.1 Distributed Interactive Computing	1
1.2 The Client Assignment Problem	3
1.3 Thesis Contributions	4
1.4 Thesis Organization	5
 2 Literature Review	 6
2.1 Architectures	6
2.2 Client Assignment	10
2.3 Server Placement	12
2.4 Consistency and Fairness	13
2.5 Summary	14
 3 Problem Formulation	 15
3.1 System Model	15
3.2 Formulation of The Client Assignment Problem for Discrete DIAs	16
3.3 Formulation of The Client Assignment Problem for Continuous DIAs	18
3.3.1 Consistency and Fairness Models	18
3.3.2 Minimum Achievable Interaction Time	19
3.3.3 Problem Statement	25
3.4 Further Considerations	25

4	Client Assignment for Discrete Distributed Interactive Applications	28
4.1	NP-Completeness Results	28
4.2	Heuristic Algorithms	36
4.2.1	Nearest-Server Assignment	36
4.2.2	Modify Assignment	40
4.2.3	Distributed-Modify Assignment	53
4.2.4	Dealing with Server Capacity Constraints	55
4.3	Experimental Evaluation	57
4.3.1	Experimental Setup	57
4.3.2	Number of Assignment Modifications	58
4.3.3	Performance Comparison of Different Algorithms	60
4.3.4	Impact of Server Capacity	61
4.3.5	Dynamic Scenarios	64
4.4	Optimal Assignment for Tree Topologies	69
4.4.1	Generalized Problem Definitions	70
4.4.2	Recurrences for Dynamic Programming	72
4.4.3	Extensions	82
4.5	Summary	84
5	Client Assignment for Continuous Distributed Interactive Applications	85
5.1	Hardness Results	85
5.2	Heuristic Algorithms	89
5.2.1	Nearest-Server Assignment	89
5.2.2	Greedy Assignment	92
5.2.3	Distributed-Modify Assignment	97
5.2.4	Dealing With Server Capacity Constraints	100
5.3	Experimental Evaluation	105
5.3.1	Experimental Setup	105
5.3.2	Performance Comparison of Different Algorithms	106
5.3.3	Impact of Server Capacity	109
5.3.4	Dynamic Scenarios	111
5.4	Summary	115

6 Conclusion	118
6.1 Contributions	119
6.2 Future Research Directions	121
6.2.1 Optimal Client Assignment for Continuous DIAs in Tree Topologies .	121
6.2.2 Server Placement	121
6.2.3 Inter-server Link Provisioning	123
6.2.4 Virtual Machine Placement in Distributed Clouds	124
Author's Publications	125
References	126

List of Figures

1.1	Distributed server architecture.	2
2.1	Example of the replication approach offering better interactivity than the partitioning approach.	9
3.1	Interaction paths in a network.	16
3.2	An example illustrating the minimum achievable interaction time.	24
4.1	Assigning client c to server s_2 cannot be optimal.	29
4.2	Assignment A_1 cannot be an optimal assignment.	30
4.3	Illustration of the instance Q of the client assignment problem.	34
4.4	The approximation ratio 3 of <i>Nearest-Server Assignment</i> is tight.	38
4.5	Unbounded approximation ratio of <i>Nearest-Server Assignment</i> when the network latency does not satisfy the triangle inequality.	39
4.6	The approximation ratio n of <i>Modify Assignment</i> and <i>Distributed-Modify Assignment</i> is tight.	44
4.7	Curves of $f(t)$ and $g(t)$ for $n = 1000$	53
4.8	An example in which changing the assigned servers of two clients simultaneously increases the total interaction path length.	54
4.9	The approximation ratio of the “capacitated” <i>Nearest-Server Assignment</i> is larger than 3.	56
4.10	Distribution of the number of assignment modifications performed by <i>Modify Assignment</i> for networks with 80 randomly placed servers.	59
4.11	Performance of <i>Distributed-Modify Assignment</i> after different numbers of stages of assignment modifications for networks with 80 servers.	59
4.12	Performance of client assignment algorithms for different numbers of servers.	62
4.13	Performance of client assignment algorithms for different server capacities.	63
4.14	The number of participating clients over the event sequence.	64

4.15	Performance of client assignment algorithms with dynamic client participation.	66
4.16	Performance of client assignment algorithms with dynamic network latency. .	68
4.17	The subtree T_u rooted at node u and the link set E_u	70
4.18	Generalized problems.	72
4.19	T_u and its subtrees.	72
4.20	A recursive solution to the (u, x) -problem.	75
4.21	A recursive solution to the (u, x, s) -problem.	77
4.22	A recursive solution to the $(u, 0)$ -problem.	80
5.1	Example of an instance of the minimum set cover problem with $K = 2$ and the corresponding distance matrix of the client assignment problem.	86
5.2	The approximation ratio 3 of <i>Nearest-Server Assignment</i> is tight.	91
5.3	Unbounded approximation ratio of <i>Nearest-Server Assignment</i> when the net- work latency does not satisfy the triangle inequality.	91
5.4	The approximation ratio of <i>Greedy Assignment</i> is asymptotically tight.	96
5.5	An example in which changing the assigned servers of two clients simultane- ously increases the maximum interaction path length.	98
5.6	Unbounded approximation ratio of <i>Distributed-Modify Assignment</i>	99
5.7	The approximation ratio 5 of the “capacitated” <i>Nearest-Server Assignment</i> is tight.	102
5.8	The approximation ratio of the “capacitated” <i>Greedy Assignment</i> is asymp- totically tight.	104
5.9	Performance of client assignment algorithms for different numbers of servers.	107
5.10	Cumulative distribution of the normalized interactivity for networks with 80 randomly placed servers.	108
5.11	Cumulative distribution of the number of assignment modifications performed by <i>Distributed-Modify Assignment</i>	108
5.12	Performance of client assignment algorithms for different server capacities. . .	110
5.13	Performance of client assignment algorithms with dynamic client participation.	114
5.14	Performance of client assignment algorithms with dynamic network latency. .	116
6.1	Example of the server placement problem.	122

List of Tables

4.1	Computation times for networks with 80 servers	61
4.2	Number of clients reassigned after three stages of assignment modifications .	69
5.1	Computation times for networks with 80 servers	109

List of Notations

V	the set of nodes in a network
C	the set of clients in a network
S	the set of servers in a network
$d(u, v)$	the distance between nodes u and v
$s_A(c)$	the server that client c is assigned to in assignment A
$D_d(A)$	the total length of interaction paths between all client pairs in assignment A
δ	the constant lag for operation execution
$\Delta_{u,v}$	the offset of node u 's simulation time relative to node v 's simulation time
$D_c(A)$	the maximum length of interaction paths between all client pairs in assignment A
$a(s)$	the number of clients assigned to server s
T_u	the subtree rooted at node u in a tree topology
$p(u)$	the parent of node u in a tree topology
E_u	the set of links incident upon the nodes in subtree T_u
$\mathcal{C}(E_u)$	the contribution of the links E_u to the total length of all interaction paths between clients
$A(u, x)$	the optimal T_u -client assignment to the (u, x) -problem
$A(u, x, s)$	the optimal T_u -client assignment to the (u, x, s) -problem
$A(u, 0)$	the optimal T_u -client assignment to the $(u, 0)$ -problem
$l(u, x)$	the value of $\mathcal{C}(E_u)$ for the optimal assignment $A(u, x)$
$l(u, x, s)$	the value of $\mathcal{C}(E_u)$ for the optimal assignment $A(u, x, s)$
$l(u, 0)$	the value of $\mathcal{C}(E_u)$ for the optimal assignment $A(u, 0)$

Chapter 1

Introduction

1.1 Distributed Interactive Computing

The steady growth in broadband Internet accesses and PC penetration is leading to a shift in emphasis from individual computers to distributed interactive computing that allows multiple participants at different locations to interact with each other in real time. Recent years have witnessed rapid development of Distributed Interactive Applications (DIAs) in many areas such as interactive digital media and entertainment [21], distributed interactive simulation [6], and collaborative computer-aided design and engineering [5].

Based on whether the application's state would change due to the passing of time, DIAs can be classified into discrete DIAs and continuous DIAs. Discrete DIAs change their states in response to user operations only. Examples of discrete DIAs include collaborative text editors [67] and collaborative computer-aided design tools [5]. Continuous DIAs change their states not only in response to user operations but also due to the passing of time. Examples of continuous DIAs include distributed virtual environments [66], distributed interactive simulations [17] and multiplayer online games [31].

In distributed interactive computing, the application's state (such as the virtual worlds in multiplayer online games and the shared workspaces in collaborative design tools) is typically maintained by servers [60, 64]. Participants, known as clients, are responsible for sending user-initiated operations to the servers. When the application's state changes, state

updates are delivered by the servers to the clients in real time to reflect the change. To enable high quality of interaction experience, user-initiated operations and state updates must be disseminated and reflected timely. For example, it has been reported that the maximum delay to maintain acceptable online gaming experience varies between 100ms to 1000ms depending on the type of the game [24]. Therefore, the interactivity performance of DIAs is important for supporting graceful interactions among participants.

Figure 1.1: Distributed server architecture.

Network latency is known as a major barrier to providing decent interactivity in DIAs [29]. Network latency cannot be eliminated from the interactions among participants and has a lower theoretical limit imposed by the speed of light. Increasing geographical spreads of participants in large-scale DIAs nowadays are making distributed server deployment vital for combating the network latency. Latency-driven distribution of servers is essential even when there are no limitations on the availability of server resources at one location [64]. In a distributed server architecture, the application's state is often replicated across a group of geographically distributed servers [12, 25]. As shown in Figure 1.1, the servers communicate with each other directly, while each client is assigned to one server through which it interacts with all the other clients. When issuing an operation, a client first sends the operation to its assigned server. Then, the server forwards the operation to all the other servers. On receiving the operation, each server calculates the new state of the application and sends a state update to all the clients assigned to it. In this thesis, we consider the distributed

server architecture in which each server maintains a full copy of the application's state.

1.2 The Client Assignment Problem

The interactivity performance of DIAs can be characterized by the duration from the time when a participant issues an operation to the time when the effect of the operation is presented to the same participant or other participants [43]. This duration is known as the *interaction time* between participants. As aforementioned, in the distributed server architecture, the clients interact with one another through their assigned servers. Thus, the interaction time between any pair of clients must include not only the network latencies between the clients and their assigned servers, but also the network latency between their assigned servers. These network latencies are directly affected by how the clients are assigned to the servers.

In addition to the network latency involved in the interaction, the interaction time may also be influenced by the consistency and fairness requirements of DIAs. Consistency means that shared common views of the application state must be created among all clients to support meaningful interactions [29]. Fairness, on the other hand, is to ensure that all clients have the same chance of participation regardless of their network conditions [14, 51]. Both consistency and fairness are important factors in providing pleasant user experiences. Maintaining consistency and fairness in DIAs usually introduces artificial synchronization delays in the interactions among clients due to diverse network latencies [14, 27, 35, 50, 56]. These synchronization delays are also dependent on the assignment of clients to servers. Therefore, how to assign the clients to the servers in DIAs is of crucial importance to their interactivity performance.

In this thesis, we investigate the problem of effectively assigning clients to servers for maximizing the interactivity of DIAs. Finding the optimal client assignment is a challenging task, even without considering any synchronization delays. An intuitive approach is to

assign each client to its nearest server [50, 75]. While this assignment reduces the client-to-server latency, it could considerably increase the latency between the assigned servers of different clients and thus make the interactivity far worse than optimum. On the other hand, assigning all clients to a single server eliminates the contribution of the interserver latency to the interaction time, but such assignment may significantly increase the latencies between the clients and their assigned server, defeating the purpose of geographical distribution of servers. An optimal client assignment for maximizing the interactivity performance of DIAs should strike a balance between the interserver latency and the client-to-server latency. Further considering the synchronization delays in the interaction would make the client assignment problem even more complicated.

1.3 Thesis Contributions

The main contributions of this thesis are summarized below:

- We mathematically model the interactivity performance of different types of DIAs, taking the consistency and fairness requirements into consideration. Based on the analysis, we formulate the client assignment problems for both discrete and continuous DIAs as combinatorial optimization problems. We prove that both problems are NP-complete via polynomial time reductions from existing NP-complete problems. Some approximability results are also obtained for the client assignment problem of continuous DIAs.
- For each client assignment problem, besides the intuitive *Nearest-Server Assignment* that assigns each client to its nearest server, several heuristic assignment algorithms are proposed, including both centralized and distributed algorithms. The approximation ratios of these algorithms are theoretically analyzed and examples are given to show the tightness of these ratios. We also adapt each proposed assignment algorithm to

deal with server capacity constraints when assuming that the servers have limited capacities.

- The performance of the assignment algorithms is experimentally evaluated using real Internet latency data. The results show that the proposed assignment algorithms produce near optimal interactivity and significantly outperform the *Nearest-Server Assignment* algorithm. We also evaluate the adaptivity of the assignment algorithms to dynamics in the networks. Our proposed *Distributed-Modify Assignment* algorithm shows good adaptivity to dynamics in both client participation and network latency.
- For the client assignment problem of discrete DIAs, we show that there exists a polynomial-time optimal solution for the special case of tree network topologies. An efficient algorithm is developed to compute the optimal client assignment.

1.4 Thesis Organization

This thesis consists of six chapters. The remainder of the thesis is organized as follows. Chapter 2 reviews the related work. Chapter 3 formulates the two client assignment problems. Chapter 4 investigates the client assignment problem for discrete DIAs. Chapter 5 studies the client assignment problem for continuous DIAs. Finally, Chapter 6 summarizes the research contributions and concludes the thesis.

Chapter 2

Literature Review

This chapter reviews the related works of resource provisioning in DIAs. The architectures that support DIAs are summarized in Section 2.1. Previous studies related to the client assignment problem are outlined in Section 2.2. Existing works on the server placement problem are discussed in Section 2.3. Finally, Section 2.4 studies the issues of consistency and fairness in DIAs.

2.1 Architectures

Three types of architectures have been proposed to support DIAs: the centralized server architecture, the peer-to-peer architecture, and the distributed server architecture. In the centralized server architecture, the application's state is maintained by a single server. Clients forward the user-initiated operations to the server. The server collects the operations from the clients, calculates the new state of the application, and sends the state updates to all clients. On receiving the state updates, the clients present the new state to the users. Systems such as Community Place [49] and MASSIVE-3 [37] are implemented in the centralized server architecture. The primary advantage of the centralized server architecture is that consistency is not an issue, since there is only one authoritative copy of the application's state that is maintained by a single server. In addition, it allows the administrator of the DIA (e.g., the gaming company) to retain full control over the application. However, since the communication between all clients goes through a central server, the centralized

server architecture would result in high interaction time for the clients that are located far away from the server to participate due to the large network latencies between these clients and the server.

In the peer-to-peer architecture, there is no central server for managing the DIA. Instead, each client is responsible for maintaining its own copy of the application's state based on the operations issued by other clients. Systems such as NPSNET [32] and DIVE [19] are implemented in the peer-to-peer architecture. Since the user operations are transmitted directly from one client to another, the interaction time between clients in the peer-to-peer architecture is reduced compared to that in the centralized server architecture. However, the distribution of the application's state makes it difficult for the DIA administrator to retain full control over the application. Moreover, to maintain consistency among all clients, each operation issued by one client should be sent to all the other clients. As a result, the client-side bandwidth consumption in the peer-to-peer architecture can be much higher than that in the centralized server architecture. Thus, the peer-to-peer architecture does not scale well with the number of clients [9, 55, 62]. Group communication and interest management techniques can be used to improve the scalability of the peer-to-peer architecture [10, 44, 61]. For example, in the interest management of multiplayer online games, each client specifies an area-of-interest (AOI) in the virtual world and maintains a partial virtual world state corresponding to its AOI. As a result, each client only needs to receive the operations in its AOI for consistency maintenance. This mechanism makes it possible to allow more users to participate in the DIA simultaneously.

The distributed server architecture has been proposed to integrate the advantages of the centralized server and peer-to-peer architectures [9, 12, 25, 40, 46, 55, 59]. In the distributed server architecture, the application's state is maintained across a group of distributed servers. Each client connects to one server for sending user-initiated operations and receiving state updates, while the servers calculate the application's state based on the

operations issued by clients. Like the centralized server architecture, the application is easy to be deployed under the centralized control of the DIA administrator. As in the peer-to-peer architecture, the transmission of operations and state updates does not need to make a detour through a central server, thereby reducing the interaction time between clients. There are two approaches for implementing the distributed server architecture: partitioning and replication.

In the partitioning approach, the application's state is divided into different parts (such as disjoint regions of the virtual worlds in multiplayer online games) and each part is assigned to and managed by one server [40, 59]. Each client connects to one server and is allowed to interact with only the clients connecting to the same server within the part of the application's state maintained at the server. Many partitioning algorithms have been proposed to dynamically divide the application's state based on its characteristics, such as the number of clients in each region of the virtual world in a multiplayer online game, to ensure approximately the same amount of workload and bandwidth requirement at each server [11, 20, 30, 48, 52, 59]. The partitioning approach improves the scalability compared to the centralized server architecture as the number of clients that each server needs to handle is reduced. In addition, since each part of the application's state is managed by only one server and clients connecting to different servers do not communicate with each other, consistency of the application's state is not an issue. However, a client interested in a particular part of the application's state may be geographically far from the server responsible for this part, which gives rise to large network latency between the client and the server and thus high interaction time for the client. Moreover, the partitioning approach also severely restricts the extent of interaction among participants.

In the replication approach, each server maintains a full copy of the application's state [12, 25]. On receiving the user-initiated operations, the servers forward the operations among themselves, and each server calculates the application's state independently based

Figure 2.1: Example of the replication approach offering better interactivity than the partitioning approach.

on the operations received. By replicating the application's state across the servers, both the scalability and the interactivity can be improved compared to the centralized server architecture. First, each server only needs to manage a portion of the clients. Thus, more clients can participate simultaneously in the DIA. In addition, regardless of the client's interest in the application's state, each client can be assigned to any server in the network, making it possible to find the most appropriate server for each client to improve the interactivity. Thus, the replication approach can offer better interactivity than the partitioning approach. Figure 2.1 presents an example, where c_1, c_2, c_3 are three clients located in the same region of the virtual world of an online game, and s_1, s_2, s_3 are three servers. Each client has a direct network link to one server, and all servers are connected to each other via network links. Assume that the network latencies on all the links are the same. In the partitioning approach, all clients must connect to the same server, regardless of how the virtual world regions are assigned to the servers. Thus, it is inevitable that the communication between two of the clients must make a detour through all the three servers as shown in Figure 2.1(a). In contrast, in the replication approach, each client can be assigned to its nearest server. Then, there is no detour in the communication between all pairs of clients (see Figure 2.1(b)). One issue in the replication approach is to maintain consistency of the

application's state among all servers, which shall be discussed in Section 2.4. In this thesis, we consider the distributed server architecture with the replication approach.

2.2 Client Assignment

The salient feature of real-time mutual interactions makes the client assignment in DIAs fundamentally different from that in web content distribution. In web content distribution, since web clients simply download contents from web servers, it is straightforward to direct web clients to their nearest servers for minimizing the access latency [26, 63]. In contrast, in DIAs supported by the distributed server architecture, each client connects to one server through which it interacts with all the other clients, and the interaction time between clients includes both the client-to-server latency and the interserver latency.

Previous studies on client assignment for interactivity enhancement of DIAs considered only the client-to-server latency as the objective of optimization. Since each client can always be assigned to its nearest server to obtain the minimum client-to-server delay, the client assignment problem is usually studied with limited server capacities [69, 70, 74]. Ta *et al.* [69, 70] studied the client assignment problem for the distributed server architecture with the partitioning approach. They formulated the problem and proved that it is NP-hard for general networks. Then, a greedy algorithm was proposed to solve the problem approximately which takes into consideration the performance loss (in term of the total client-to-server latency) when the regions of the virtual world cannot be assigned to their nearest servers due to insufficient server capacity. Webb *et al.* [74] studied the client assignment problem for the distributed server architecture with the replication approach and obtained a similar formulation to that of [69, 70]. When assuming that each client consumes the same amount of server capacity, this client assignment problem can be solved in polynomial time by employing the sequential assignment algorithm [72] of the terminal

assignment problem. Nevertheless, as shall be shown by our experimental results, reducing the client-to-server latency alone is not effective for improving the interactivity of DIAs.

When synchronization delays for maintaining consistency and fairness are not considered in the interaction times, the client assignment problem for DIAs is similar to that of the fixed-hub single allocation problem that arose in transportation systems. In these systems, hubs are placed in the network and used as transshipment points. The traffic in the network may flow between the non-hub nodes through hubs and/or between hubs. The fixed-hub single allocation problem asks: given a set of fixed hubs, how to assign each node to one hub to minimize the total cost of traffic routing in the network. Campbell [18] proposed two heuristic solutions called *MAXFLO* (which we shall include in our evaluation in Chapter 4) and *ALLFLO* (which is an exponential time algorithm). Iwasa *et al.* [42] presented a 3-approximation algorithm that is similar to the intuitive *Nearest-Server Assignment* algorithm. They further developed a 2-approximation algorithm by using a linear programming relaxation and a randomized rounding technique. However, the size of the linear programming relaxation is large and thus the algorithm is only suitable for small networks. In addition, it is a centralized algorithm that is difficult to deal with dynamic scenarios in practice. In this thesis, we shall propose distributed algorithms that can incrementally adapt the client assignment to dynamics in client participation and network latency.

The client assignment problem for DIAs is also to some extent similar to that of the NP-hard quadratic assignment problem (QAP), which aims to allocate a set of facilities to a set of locations such that the total cost is minimized [16]. In the QAP, the cost for each pair of facilities is the product of the flow between the two facilities and the distance between their assigned locations, while the cost in our client assignment problem is the length of the three-segment interaction path between each pair of clients which includes not only the interserver path but also the client-to-server paths. In addition, each location in the QAP can be allocated one facility only, i.e., it is a one-to-one mapping from the facilities to the

locations. This differs from our client assignment problem in that each client is free to be assigned to any server.

2.3 Server Placement

Another issue relevant to interactivity enhancement is server placement, which aims to find where to place servers in the network [26, 50]. Server placement is often related to several graph theoretic problems including the facility location problem and the k -center and k -median problems. These problems are all NP-hard and have been well studied in the literature. The goal of the facility location problem is to find the locations to place facilities so as to minimize the sum of the distances from a given set of points to their nearest facilities and the costs of opening the facilities. Hochbaum [39] first gave a greedy algorithm with an approximation ratio of $O(\log n)$. Korupolu *et al.* [45] proposed a local search heuristic that has a constant approximation ratio of $5 + \varepsilon$, which has been subsequently improved to 3 [8]. Aggarwal *et al.* [4] presented a 3-approximation algorithm assuming that each facility has the same limited service capacity. The k -median problem aims to place a given number of k servers in the network to minimize the total distance from the clients to their closest servers. It has been shown that polynomial-time optimal k -median solutions exist on tree graphs [71]. For general graphs, a naive greedy algorithm has an approximation ratio of $\Omega(n)$ [23]. Local search heuristics have been proposed with the approximation ratio of $3 + \varepsilon$ [8]. The k -center problem targets at placing k servers in the network to minimize the maximum distance between a client and its closest server. There is a 2-approximation k -center algorithm and this ratio cannot be further improved unless $P = NP$ [36]. Different from the above work, Lee *et al.* [50] studied how to place servers for online games taking into account the interserver latency. However, no approximation ratio was given for their proposed algorithm. Choi *et al.* [22] studied the session server placement problem, in which the interactions between clients are represented by sessions and only clients involved in the

same session communicate with each other. One client can be involved in several different sessions. The objective of the session server placement problem is to minimize the total latency between the clients and the servers in all sessions. The session server placement problem can be formulated as a k -median problem for general networks. A polynomial-time optimal solution exists for line networks for a special case in which each session contains only two clients. De Vleeschauwer *et al.* [28] studied a server placement problem similar to [22] for multiplayer online games. For each pair of clients, at least one server is required through which one client interacts with the other. The objective is to minimize the weighted sum of the total network latency and the number of servers placed in the network. Two heuristic algorithms were proposed to solve the problem, but no approximation ratio was guaranteed.

Server placement is normally planned on long-term basis as implementing new placement solutions often involves amendment to hardware deployment or lease agreements with third-party service providers. Client assignment complements server placement in that it aims to assign clients to appropriate servers given a set of servers placed. Since client assignment deals with only software connections between clients and servers, it can be adjusted promptly to adapt to dynamics in the networks. Our work in this thesis shows that finding an optimal client assignment is a challenging task, even under carefully planned server placements.

2.4 Consistency and Fairness

Consistency and fairness are two important requirements for DIAs. A number of synchronization mechanisms have been developed for consistency maintenance. They can be classified into optimistic and pessimistic mechanisms [29]. In optimistic mechanisms such as TimeWarp [56, 58], all servers execute operations immediately after receiving them, and the application states are repaired when inconsistency is detected. The repairs, however, may produce consistency-related artifacts that greatly disturb the user behavior. Pessimistic

mechanisms such as bucket synchronization [35] and local lag [56] add artificial synchronization delays before operation execution to reduce the chance for inconsistency and artifacts to arise. Due to the artificial synchronization delays, pessimistic mechanisms increase the interaction time between clients compared to optimistic mechanisms. In pessimistic mechanisms, the interaction time between two clients could be longer than the actual network latency between them.

For the fairness requirement, some studies used the variation of the time for each client to see the effect of its own operation as the metric of fairness [13, 15, 33, 76]. This metric, however, does not guarantee that all operations are executed in a fair order, i.e., in the order of their issuances by the clients. Some other work [50, 51] studied the required amount of delay to execute operations in the fair order at a single server. However, these results cannot be directly generalized to distributed servers. To the best of our knowledge, there has been no work on modeling the minimum achievable interaction time of DIAs under the consistency and fairness requirements.

2.5 Summary

The client assignment problem is concerned with assigning the clients to a given set of servers, which complements the server placement problem that determines where to place servers in the network. Previous work on client assignment for enhancing the interactivity performance of DIAs aimed to reduce the client-to-server latency only. Different from existing research, the objective of our study is to directly minimize the end-to-end interaction time between clients, taking into consideration possible synchronization delays that are required to maintain consistency and fairness in the DIA. As far as we know, we are the first to address the client assignment problem with mathematical modeling of the interaction times among clients in DIAs. We investigate the client assignment problem under a variety of consistency and fairness constraints in this thesis.

Chapter 3

Problem Formulation

This chapter formulates the client assignment problems for DIAs. We first introduce the system model and provide some basic definitions in Section 3.1. Then, the client assignment problems for discrete DIAs and continuous DIAs are formulated in Sections 3.2 and 3.3 respectively.

3.1 System Model

We model the underlying network supporting the DIA by a graph consisting of a set of nodes V . A distance $d(u, v) > 0$ is associated with each pair of different nodes $(u, v) \in V \times V$, representing the network latency of the routing path between nodes u and v . We define that $d(v, v) = 0$ for each node $v \in V$. Consider a distributed server architecture. Let $S \subseteq V$ be the set of servers in the network and $C \subseteq V$ be the set of clients. Each client needs to be assigned to a server for sending user operations and receiving state updates. An assignment A is a mapping from C to S . For each client $c \in C$, we denote by $s_A(c) \in S$ as the server that client c is assigned to.

The clients interact with one another through their assigned servers. Specifically, when a client c_i issues an operation, the effect of the operation is presented to another client c_j through the following process. First, c_i sends the operation to its assigned server $s_A(c_i)$. Then, $s_A(c_i)$ forwards the operation to c_j 's assigned server $s_A(c_j)$ if they are different. Finally, $s_A(c_j)$ executes the operation, possibly after some artificial synchronization delay,

Figure 3.1: Interaction paths in a network.

and then delivers the resultant state update to c_j . In the above interaction process, the paths from c_i to $s_A(c_i)$, from $s_A(c_i)$ to $s_A(c_j)$, and from $s_A(c_j)$ to c_j are involved. Similarly, if c_j issues an operation, the same three paths in the network are involved in the interaction process for c_i to see the effect of the operation. Therefore, we refer to the concatenation of these three paths as the *interaction path* between c_i and c_j . For example, in the network of Figure 3.1, suppose clients c_1 and c_2 are assigned to server s_1 , and client c_3 is assigned to server s_2 . Then, the interaction path between c_1 and c_2 is indicated by the dotted line, and the interaction path between c_1 and c_3 is indicated by the dashed line. As can be seen, the interaction path between two clients does not necessarily follow the shortest path between them.

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

3.2 Formulation of The Client Assignment Problem for Discrete DIAs

For discrete DIAs such as collaborative editing and design systems, it is generally not necessary to guarantee that the application's state is always the same at all the servers at

any point in time [67]. A user operation can be executed by different servers at different times and in different orders. However, the servers, after executing the same set of user operations, are expected to have the same states. Since the orders of operation execution may not be identical at different servers, consistency is not automatically guaranteed among the application's states at different servers. Existing approaches for establishing consistency in discrete DIAs are mainly to ensure that the application's states at all the servers are eventually the same as if all these operations had been executed in a particular order based on the casual relationships among them [67, 68].

To minimize the interaction time, it is intuitive that each user operation should be executed immediately by each server upon its arrival at the server, and after that, the server sends the resultant state updates to its clients right away. When inconsistency due to diverse execution orders is detected, consistency maintenance techniques such as operational transformation [68] can be employed to repair the application's state. Since no artificial synchronization delay is introduced in the interaction, the interaction time between two clients c_i and c_j can be simply considered as the total network latency along their interaction path, i.e., the length of their interaction path. We use the average interaction time between all pairs of clients as a measure of the overall interactivity of the DIA. The average interaction time indicates how long it takes for the effect of an operation to be presented to a participant on an average level. We consider all unordered pairs of clients (c_i, c_j) , where c_i can be the same as c_j . Given a set of clients C , there are a total of $|C|(|C| + 1)/2$ unordered client pairs. Hence, the average interaction time is given by

$$\frac{2}{|C|(|C| + 1)} \sum_{c_i, c_j \in C} (d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)).$$

Note that the total number of client pairs is fixed for a given set of clients. Thus, to minimize the average interaction time, it is equivalent to minimize the total length of interaction paths between all client pairs. Therefore, the client assignment problem for discrete DIAs is formulated as follows.

Definition 3.1 (Client Assignment Problem for Discrete DIAs) *Given a set of servers S and a set of clients C in a network, find a client assignment A that minimizes the total length of interaction paths between all client pairs, i.e.,*

$$\text{minimize } D_d(A) = \sum_{c_i, c_j \in C} \{d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)\}.$$

Although the above formulation assumes that interaction exists between each pair of clients, our analysis and proposed algorithms in Chapter 4 can be easily generalized to handle the situation where each client only interacts with a portion of the other clients or to reflect different amounts of interaction between different pairs of clients.

3.3 Formulation of The Client Assignment Problem for Continuous DIAs

3.3.1 Consistency and Fairness Models

In continuous DIAs, the progress of the application state is typically measured by the time elapsed since the initial state of the application [56]. We shall call it the *simulation time*. For instance, the simulation time of a multiplayer online game records the time elapsed in its virtual world. In the distributed server architecture, each server and client has a copy of the application state and its associated simulation time. The simulation times of all servers and clients should advance at the same rate, which is often the same as that of the wall-clock time in order to provide realistic real-time interaction experiences. However, they do not have to be synchronized, i.e., their readings do not have to be the same at the same wall-clock time. Normally, the simulation time of a client lags behind the simulation time of its assigned server due to the network latency of delivering state updates from the server to the client.

The consistency requirement for continuous DIAs is to ensure that all clients share the same view of the application state when their respective simulation times reach the same

value. This is automatically guaranteed among the clients assigned to the same server because they all inherit the application state from their assigned server through state updates. Nevertheless, the application states seen by the clients assigned to different servers may not be identical at the same simulation time if the application states maintained by their assigned servers are not consistent. Since the state of a continuous DIA changes due to both user operations and time passing, to ensure consistency among the application states at the servers, each user operation must be executed by all servers at the same simulation time.

The fairness requirement is to ensure that all clients have the same chance of participation regardless of their network conditions. This is particularly important for applications where users compete with each other. In essence, fairness is concerned with the order of executing user operations [51]. For example, a participant would gain an unfair advantage in an air combat game if an action performed by him takes effect before another action performed earlier by a different participant. To guarantee fairness in continuous DIAs, the order of operation execution must be the same as the operation issuance order at the clients based on the simulation time. In addition, the time interval between the issuances of two operations in terms of simulation time must also be preserved between the executions of these operations. This entails executing each operation at the server at a simulation time that is of a constant lag behind the simulation time of the operation issuance.

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

3.3.2 Minimum Achievable Interaction Time

Let δ denote the constant lag for operation execution. We first show that the average interaction time between all client pairs is equal to δ . For each pair of nodes $u, v \in C \cup S$ in

the network, we denote by $\Delta_{u,v}$ the offset of node u 's simulation time relative to node v 's simulation time (a positive offset means that u 's simulation time is ahead of v 's simulation time). Since the simulation times of all servers and clients advance at the same rate, the offset $\Delta_{u,v}$ for each node pair u, v is a constant. Obviously, we have $\Delta_{u,v} + \Delta_{v,u} = 0$, and $\Delta_{u,v} + \Delta_{v,w} = \Delta_{u,w}$.

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

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

- (i) When any client issues an operation at its simulation time t , all servers are able to receive the operation before their respective simulation times reach $t + \delta$.
- (ii) All clients are able to receive the resultant state update of executing the operation before their respective simulation times reach $t + \delta$.

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

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

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

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

$$\forall c, s, \quad d(c, s_A(c)) + d(s_A(c), s) + \Delta_{s,c} \leq \delta. \quad (\text{Eq. 3.1})$$

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

$$\forall c, \quad d(s_A(c), c) + \Delta_{c,s_A(c)} \leq 0. \quad (\text{Eq. 3.2})$$

We now derive the minimum possible value of δ from inequalities (Eq. 3.1) and (Eq. 3.2). For any client pair c_i and c_j , based on (Eq. 3.1), we have

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

Based on (Eq. 3.2), we have

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

By adding up the above two inequalities, it follows that

$$2d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + \Delta_{s_A(c_j), s_A(c_i)} \leq \delta. \quad (\text{Eq. 3.3})$$

Similarly, it can be deduced that

$$2d(c_j, s_A(c_j)) + d(s_A(c_j), s_A(c_i)) + \Delta_{s_A(c_i), s_A(c_j)} \leq \delta. \quad (\text{Eq. 3.4})$$

By adding up (Eq. 3.3) and (Eq. 3.4), we have

$$d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(c_j, s_A(c_j)) \leq \delta. \quad (\text{Eq. 3.5})$$

The expression on the left side of inequality (Eq. 3.5) is the length of the interaction path between clients c_i and c_j . Since c_i, c_j can be any pair of clients, (Eq. 3.5) indicates that δ should be no less than the maximum length of interaction paths between all client pairs, which is given by

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

Finally, we show that $\delta = D$ can be achieved by properly setting the offsets between simulation times at the servers and clients. We first synchronize the simulation times at all clients, i.e., setting $\Delta_{c, c'} = 0$ for all client pairs c, c' . Then, the values of $\Delta_{s, c}$ between a given server s and all clients c are set as follows:

$$\Delta_{s, c} = D - \max_{c' \in C} \{d(c', s_A(c')) + d(s_A(c'), s)\}, \quad (\text{Eq. 3.6})$$

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

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

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

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

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

We present an example to illustrate the minimum achievable interaction time. In the network shown in Figure 3.2(a), three clients c_1, c_2, c_3 are assigned to servers s_1, s_2, s_3 respectively. A straightforward strategy of time setting is to make the simulation times at all clients synchronized, and also make the simulation times at all servers synchronized (Figure 3.2(b)). The simulation time of clients must lag behind the simulation time of servers due to the network latency of delivering state updates. Suppose that at simulation time 0, one of the three clients issues an operation. If c_1 issues the operation, its operation first reaches server s_1 at simulation time 6 as shown in Figure 3.2(b). Then, the operation is forwarded to the other two servers, which receive it at simulation time 7. Figure 3.2(b) also shows the delivery of the operation if it is issued by client c_2 or c_3 . As can be seen, the latest possible simulation time for a server to receive the operation is 11 (i.e., the time for server s_2 to receive the operation if it is issued by c_3). Therefore, the constant lag for operation execution must be at least 11. If the operation is executed at simulation time 11, all the clients receive and display the resultant state update at simulation time 11, and consequently, the interaction time is 11. On the other hand, if the offset between each server and all clients is set according to (Eq. 3.6), the delivery of operation and state update is

(a) Network topology

(b) Interaction time when all servers are synchronized in simulation time

(c) Minimum achievable interaction time

Figure 3.2: An example illustrating the minimum achievable interaction time.

shown in Figure 3.2(c). It can be seen that regardless of its origin, the operation can be received and executed by all servers by simulation time 9. Thus, the resultant interaction time is 9, which equals the maximum interaction path length.

3.3.3 Problem Statement

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

Definition 3.2 (Client Assignment Problem for Continuous DIAs) *Given a set of servers S and a set of clients C in a network, find a client assignment A that minimizes the maximum length of interaction paths between all client pairs, i.e.,*

$$\text{minimize } D_c(A) = \max_{c_i, c_j \in C} \{d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j)\}.$$

3.4 Further Considerations

In this thesis, we focus on reducing the network latencies and the artificial synchronization delays involved in the interaction between clients. Thus, the above problem formulations have not taken into consideration the processing delays at the servers. In general, the processing delays at the servers are easier to improve than the network latencies [26]. A busy server can always be better provisioned (e.g., by forming a server cluster) to meet the capacity requirements and reduce the processing delay. We shall discuss in Chapters 4 and 5 how to deal with server capacity constraints in our proposed client assignment algorithms if server capacities are limited.

There may exist jitter in the network. Jitter refers to the variability of network latency. In the presence of jitter, longer synchronization delay would be required in continuous DIAs to cater for the variation in network latency in order to guarantee consistency and fairness. Our formulation of the client assignment problem for continuous DIAs is also valid in dealing with network jitter in that the distance $d(u, v)$ between each node pair u, v can be set to

any percentile of the network latency to cater for its variability to a required extent. At one extreme, setting $d(u, v)$ to the maximum possible network latency between nodes u and v would guarantee that each operation is received by all servers before its execution, thereby ensuring consistency and fairness. But this strategy may considerably degrade interactivity at large jitter. Thus, a real-world system often models a certain high percentile (e.g., 90th percentile) of the network latency to significantly reduce the chance for inconsistency and unfairness to arise [27, 51]. When inconsistency does occur due to jitter, the application state can be repaired using synchronization mechanisms such as timewarp [56] and Trailing State Synchronization (TSS) [27]. Repairing the application state, however, may create artifacts that disturb the user behavior. For instance, an artifact in an online game could mean that an opponent that has been beaten in a fight stands up again and continues to fight. Therefore, the extent to which the variability of network latency is catered reflects a trade-off among interactivity, consistency and fairness. Selecting an appropriate percentile of the network latency to model based on the application needs is beyond the scope of this thesis.

For some continuous DIAs that are fast-paced and very latency sensitive, it is possible that to guarantee good interactivity is more important than to maintain strict consistency in the system [27, 65]. Thus, optimistic synchronization mechanisms (discussed in Section 2.4) may be employed by these applications so that operations are executed immediately upon receipt at the servers. Therefore, our problem formulation for discrete DIAs is also applicable to such continuous DIAs.

We do not limit our research to $S \cap C = \emptyset$, where S and C are the server and client sets in the network. When a server and a client are located in close proximity, they may be co-located at a single node in a network model of relatively coarse granularity in which each node represents a subnet. Recent studies have also started to consider distributing "light weight" servers to the edges of the network for green computing purpose [73]. In addition,

by not restricting $S \cap C = \emptyset$, our analysis and solutions may also be applied to selecting which clients to take up the role of operation execution in the peer-to-peer architecture in which the clients are responsible for executing operations and maintaining the application state by themselves (Section 2.1).

Chapter 4

Client Assignment for Discrete Distributed Interactive Applications

This chapter studies the client assignment problem for discrete DIAs (Definition 3.1). Section 4.1 proves the NP-completeness results. Then, we propose three heuristic assignment algorithms and analyze their approximation ratios in Section 4.2. The experimental setup and results are discussed in Section 4.3. Section 4.4 presents a polynomial-time optimal solution for tree network topologies. Finally, Section 4.5 summarizes this chapter.

4.1 NP-Completeness Results

In this section, we investigate the hardness of the client assignment problem for discrete DIAs. We start by studying the characteristics of an optimal client assignment, as shown by the following two theorems.

Theorem 4.1 *Under shortest path routing, if a server s_1 is located on the shortest path between a client c and another server s_2 , then any assignment assigning c to s_2 cannot be an optimal assignment.*

Proof Suppose A is an assignment in which c is assigned to s_2 . Based on assignment A , we construct a new assignment A' by changing the assigned server of c from s_2 to

Figure 4.1: Assigning client c to server s_2 cannot be optimal.

s_1 , and keeping the assigned servers of other clients unchanged, i.e., $s_{A'}(c) = s_1$, and $\forall c_j \neq c, s_{A'}(c_j) = s_A(c_j)$. To simplify presentation, we denote by $d_A(c_i, c_j)$ and $d_{A'}(c_i, c_j)$ the lengths of the interaction paths between clients c_i and c_j under client assignments A and A' respectively, i.e.,

$$d_A(c_i, c_j) = d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + d(s_A(c_j), c_j),$$

and

$$d_{A'}(c_i, c_j) = d(c_i, s_{A'}(c_i)) + d(s_{A'}(c_i), s_{A'}(c_j)) + d(s_{A'}(c_j), c_j).$$

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

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

Under shortest path routing, the distance $d(u, v)$ satisfies the triangle inequality. So, we have

$$d(s_1, s_A(c_j)) \leq d(s_1, s_2) + d(s_2, s_A(c_j)).$$

Therefore,

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

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

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

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

Hence, the theorem is proven. $\square$

Theorem 4.2 *Under shortest path routing, if the shortest path between a client c_1 and a server s_1 and the shortest path between a client c_2 and a server s_2 have at least one common node u , then any assignment assigning c_1 and c_2 to s_1 and s_2 respectively cannot be an optimal assignment.*

Figure 4.2: Assignment A_1 cannot be an optimal assignment.

Proof As shown in Figure 4.2, suppose in an assignment A_1 , c_1 is assigned to s_1 and c_2 is assigned to s_2 . We construct two new assignments A_2 and A_3 based on A_1 . A_2 assigns both c_1 and c_2 to s_1 and keeps the assigned servers of other clients unchanged. A_3 assigns both c_1 and c_2 to s_2 and keeps the assigned servers of other clients unchanged. We show that at least one assignment of A_2 and A_3 is better than A_1 .

Denote by $d_{A_1}(c_i, c_j)$, $d_{A_2}(c_i, c_j)$, and $d_{A_3}(c_i, c_j)$ the lengths of the interaction paths between clients c_i and c_j under client assignments A_1 , A_2 , and A_3 respectively. Then, the

total length of interaction paths in assignment A_1 is

$$\begin{aligned} D_d(A_1) &= \sum_{c_i, c_j \in C} d_{A_1}(c_i, c_j) \\ &= \sum_{c_j \neq c_2} d_{A_1}(c_1, c_j) + \sum_{c_j \neq c_1} d_{A_1}(c_2, c_j) + d_{A_1}(c_1, c_2) + \sum_{c_i, c_j \neq c_1, c_2} d_{A_1}(c_i, c_j). \end{aligned} \quad (\text{Eq. 4.1})$$

By analyzing the first term of (Eq. 4.1), we have

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

Similarly, for the second term of (Eq. 4.1), we have

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

Also note that $d_{A_1}(c_1, c_2) = d(c_1, s_1) + d(s_1, s_2) + d(c_2, s_2)$. Thus, (Eq. 4.1) becomes

$$\begin{aligned} D_d(A_1) &= \sum_{c_j \neq c_1, c_2} (d(s_1, s_{A_1}(c_j)) + d(s_{A_1}(c_j), c_j)) + \sum_{c_j \neq c_1, c_2} (d(s_2, s_{A_1}(c_j)) + d(s_{A_1}(c_j), c_j)) \\ &\quad + (|C| + 1)d(c_1, s_1) + (|C| + 1)d(c_2, s_2) + d(s_1, s_2) + \sum_{c_i, c_j \neq c_1, c_2} d_{A_1}(c_i, c_j). \end{aligned} \quad (\text{Eq. 4.2})$$

Similarly, the total lengths of interaction paths in assignments A_2 and A_3 are given by

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

and

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

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

Thus, we have

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

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

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

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

Hence, the theorem is proven. $\square$

Based on Theorem 4.2, we obtain the following corollary.

Corollary 4.1 *Under shortest path routing, if a client c_2 is located on the shortest path between another client c_1 and a server s_1 , then any assignment assigning c_1 to s_1 and assigning c_2 to a different server cannot be an optimal assignment.*

Making use of the above theorems, we show that the client assignment problem for discrete DIAs (Definition 3.1) is NP-complete.

Theorem 4.3 *The client assignment problem for discrete DIAs is NP-complete.*

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

We show that the client assignment problem is NP-complete by a polynomial time reduction from the partition problem which is known to be NP-complete [34]. The partition problem is defined as follows: Given a finite set of positive integers B , does there exist a subset $B' \subseteq B$ such that $\sum_{b \in B'} b = \sum_{b \in B-B'} b$?

Let P be an instance of the partition problem in which the set B has n elements: $B = \{b_1, b_2, \dots, b_n\}$, and $\sum_{i=1}^n b_i = S$. We first construct a network with $2n$ servers $s_1, s_2, \dots, s_{2n}$ and n groups of clients $G_1, G_2, \dots, G_n$, as shown in Figure 4.3. The servers are divided into two sets $U_1 = \{s_1, s_2, \dots, s_n\}$ and $U_2 = \{s_{n+1}, s_{n+2}, \dots, s_{2n}\}$ with equal cardinality n . Each server in U_1 is connected to all servers in U_2 so that all the servers and interserver links form a bipartite graph. Each client group G_i contains b_i clients. So, there are a total of $\sum_{i=1}^n b_i = S$ clients. In each client group, one client is designated as the center and the remaining clients are connected to the center to form a star graph. The center r_i of group G_i is connected to server s_i in U_1 and server s_{n+i} in U_2 . An instance Q of the client assignment problem is then constructed on the network by setting the length of every link to 1, setting the bound $K = \frac{11}{4}S^2 - (n-2)S - n - \sum_{i=1}^n b_i^2$, and assuming that messages are routed along shortest paths in the network. It is obvious that instance Q is constructed in time polynomial to the size of instance P . In the following, we show that there exists a subset B' such that $\sum_{b \in B'} b = \sum_{b \in B-B'} b$ for instance P if and only if there exists a client assignment such that its total length of interaction paths is at most K for instance Q . The above value of K here is deliberately selected to ensure the correctness of the reducibility.

In the network that we construct, servers s_i and s_{n+i} are on the shortest paths between the center client r_i and all other servers. According to Theorem 4.1, each center client r_i must be assigned to either s_i or s_{n+i} in an optimal assignment. In addition, each center client is on the shortest paths between all the other clients in the same group and all servers. According to Corollary 4.1, all clients in one group must be assigned to the same server in an

Figure 4.3: Illustration of the instance Q of the client assignment problem.

optimal assignment. So, we only consider such assignments that assign all clients in group G_i to either server s_i in U_1 or server s_{n+i} in U_2 .

Since the length of every link is 1, the total path length in an assignment can be calculated by adding up how many times each link appears in the interaction paths between all client pairs. We divide all links in the network into three types (see Figure 4.3): the x -type links between the center client and the remaining clients in the same group, the y -type links between the center clients and their directly connected servers, and the z -type links between servers. The occurrences of x -type and y -type links in interaction paths are independent of the client assignment. Each x -type link is associated with one client that is not a center client. The x -type link contributes twice to the interaction path between this client and itself and contributes once to the interaction path between this client and each of the other clients in the network. Since there are $(S - n)$ clients that are not the center clients, all x -type links contribute a total of $(S - n)(S + 1)$ times. On the other hand, the interaction path between each pair of clients contains two y -type links. Thus, all y -type links contribute $S(S + 1)$ times in total.

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

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

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

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

$$\begin{aligned} & (S - n)(S + 1) + S(S + 1) + \sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) \\ & + 2 \cdot \sum_{\substack{s_i, s_j \in U_1 \\ s_i \neq s_j}} a(s_i) a(s_j) + 2 \cdot \sum_{\substack{s_i, s_j \in U_2 \\ s_i \neq s_j}} a(s_i) a(s_j) \\ & = (S - n)(S + 1) + S(S + 1) + \sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) \\ & + \left(\sum_{s_i \in U_1} a(s_i) \right)^2 - \sum_{s_i \in U_1} a(s_i)^2 + \left(\sum_{s_i \in U_2} a(s_i) \right)^2 - \sum_{s_i \in U_2} a(s_i)^2 \\ & = (S - n)(S + 1) + S(S + 1) + \left(\sum_{s_i \in U_1 \cup U_2} a(s_i) \right)^2 \\ & - \sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) - \sum_{s_i \in U_1 \cup U_2} a(s_i)^2. \end{aligned}$$

Since each client group G_i contains b_i clients, for servers $s_i \in U_1$ and $s_{n+i} \in U_2$, either $a(s_i) = b_i$ and $a(s_{n+i}) = 0$, or $a(s_i) = 0$ and $a(s_{n+i}) = b_i$. Therefore, for each possible assignment, $\sum_{s_i \in U_1} a(s_i)$ and $\sum_{s_i \in U_2} a(s_i)$ correspond to the sums of subsets B' and $B - B'$ respectively for a partitioning of B , and vice versa. It also follows that

$$\sum_{s_i \in U_1 \cup U_2} a(s_i) = \sum_{i=1}^n b_i = S,$$

$$\sum_{s_i \in U_1 \cup U_2} a(s_i)^2 = \sum_{i=1}^n b_i^2,$$

and

$$\sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) \leq \frac{S^2}{4}.$$

Thus, the total length of interaction paths has a lower bound

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

The lower bound K is achieved only when

$$\sum_{s_i \in U_1} a(s_i) \cdot \sum_{s_i \in U_2} a(s_i) = \frac{S^2}{4},$$

i.e.,

$$\sum_{s_i \in U_1} a(s_i) = \sum_{s_i \in U_2} a(s_i) = \frac{S}{2}.$$

Hence, the theorem is proven. $\square$

4.2 Heuristic Algorithms

A brute-force solution to the client assignment problem is computationally expensive. There are a total of $|S|^{|C|}$ different assignment strategies for an exhaustive search, where $|S|$ is the number of servers and $|C|$ is the number of clients. The search space is huge even for small values of $|S|$ and $|C|$.

In this section, we present and analyze three heuristic assignment algorithms. These heuristic algorithms do not assume any particular routing strategy and triangle inequality in the network. They compute client assignments based simply on the network latencies between different nodes, which can be obtained with existing tools like ping and King [38].

4.2.1 Nearest-Server Assignment

Nearest-Server Assignment is an intuitive algorithm that assigns each client to its nearest server. It is easy to implement and can be performed in a distributed manner because each client selects its server independently. The complexity for each client to compute its nearest server is $O(|S|)$.

The assumption of the triangle inequality is commonly made when theoretically analyzing the performance of the approximation algorithms in facility location problems [45].¹ When assuming that the network latency satisfies the triangle inequality, *Nearest-Server Assignment* has an approximation ratio of 3.

Theorem 4.4 *Nearest-Server Assignment has an approximation ratio of 3 for networks with the triangle inequality.*

Proof This proof is in spirit similar to a proof presented in [42]. Consider two different clients u and v in the network. Let n_u and n_v be the nearest servers of clients u and v , respectively. Suppose in an optimal assignment A^* , client u is assigned to a server s_u and client v is assigned to a server s_v . We immediately have the following results

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

Therefore, by the triangle inequality, we have,

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

and

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

The length of the interaction path between u and v in *Nearest-Server Assignment* is given by

$$d(u, n_u) + d(n_u, n_v) + d(v, n_v).$$

By the triangle inequality, $d(n_u, n_v)$ should be shorter than the concatenation of the paths

¹We count the number of triples (u, v, w) that satisfy the constraint $d(u, v) + d(u, w) \geq d(v, w)$ in the Internet latency data set used in our experimental evaluation (Sections 4.3 and 5.3) and observe that 92.6% of the triples satisfy the triangle inequality.

from n_u to s_u , from s_u to s_v , and from s_v to n_v . Thus, it follows that

$$\begin{aligned}
 d(u, n_u) + d(n_u, n_v) + d(v, n_v) &\leq d(u, n_u) + d(n_u, s_u) + d(s_u, s_v) + d(n_v, s_v) + d(v, n_v) \\
 &\leq d(u, s_u) + 2d(u, s_u) + d(s_u, s_v) + 2d(v, s_v) + d(v, s_v) \\
 &= 3d(u, s_u) + d(s_u, s_v) + 3d(v, s_v) \\
 &\leq 3\left(d(u, s_u) + d(s_u, s_v) + d(v, s_v)\right),
 \end{aligned}$$

where $d(u, s_u) + d(s_u, s_v) + d(v, s_v)$ is the interaction path length between u and v in the optimal assignment. Therefore, for each pair of clients, the length of their interaction path in *Nearest-Server Assignment* is within three times of that in an optimal assignment. As a result, the total length of all interaction paths in *Nearest-Server Assignment* is within three times of that in an optimal assignment. Hence, the theorem is proven. $\square$

Figure 4.4: The approximation ratio 3 of *Nearest-Server Assignment* is tight.

The approximation ratio of 3 is tight for *Nearest-Server Assignment*. Figure 4.4 presents an example² where the ratio between the total interaction path lengths in *Nearest-Server Assignment* and in an optimal assignment can be arbitrarily close to 3. Here, $c_1, c_2, \dots, c_n$ are clients and $s, s_1, s_2, \dots, s_n$ are servers. The distance between each client and server s is a , and for each i , the distance between client c_i and server s_i is $a - \varepsilon$, where $0 < \varepsilon < a$. Thus, in *Nearest-Server Assignment*, each client c_i is assigned to server s_i . The length of the interaction path between each pair of different clients is $6a - 4\varepsilon$, and the length of the

²The graphic illustrations in this thesis are meant to show pair-wise network latencies between nodes. They do not necessarily represent the physical network topologies.

Figure 4.5: Unbounded approximation ratio of *Nearest-Server Assignment* when the network latency does not satisfy the triangle inequality.

interaction path from each client to itself is $2a - 2\varepsilon$. Since there are n clients, the total length of all interaction paths in *Nearest-Server Assignment* A_N is given by

$$D_d(A_N) = \frac{n(n-1)}{2} \cdot (6a - 4\varepsilon) + n \cdot (2a - 2\varepsilon).$$

On the other hand, if the value of ε is very small, the optimal assignment A^* should assign all clients to server s , and has the total interaction path length of

$$D_d(A^*) = \frac{n(n-1)}{2} \cdot 2a + n \cdot 2a.$$

As the number of clients, n , goes towards infinity and ε approaches 0, the ratio between $D_d(A_N)$ and $D_d(A^*)$ approaches

$$\lim_{n \rightarrow \infty} \lim_{\varepsilon \rightarrow 0} \frac{D_d(A_N)}{D_d(A^*)} = \lim_{n \rightarrow \infty} \frac{n(n-1) \cdot 3a + n \cdot 2a}{n(n-1) \cdot a + n \cdot 2a} = 3.$$

This shows that the ratio between $D_d(A_N)$ and $D_d(A^*)$ can be made arbitrarily close to 3.

In the absence of the triangle inequality [53], *Nearest-Server Assignment* has unbounded approximation ratio. Figure 4.5 shows an example network containing two clients c_1, c_2 and four servers s_1, s_2, s_3, s_4 . Suppose that $r > 8$. The total interaction path length in *Nearest-Server Assignment* is $2 + 2 + (r + 2) = r + 6$, whereas the total interaction path length in the optimal assignment is $4 + 4 + 6 = 14$. Thus, the ratio between the total interaction path lengths in *Nearest-Server Assignment* and in the optimal assignment can be made arbitrarily large as r goes towards infinity.

4.2.2 Modify Assignment

Modify Assignment is a local search heuristic. It starts with an existing assignment and continues to modify the assignment in a greedy manner for reducing the total length of interaction paths. To perform an assignment modification, *Modify Assignment* examines all clients to find whether reassigning any client to a different server would reduce the total length of interaction paths. If there exist such reassignment options, the algorithm then selects the option that produces the maximum reduction and reassigns the client. *Modify Assignment* repeats the above process until the total length of interaction paths cannot be further reduced by reassigning any client. Since the assignment modifications can only improve interactivity, the resultant assignment cannot be worse than the initial assignment.

When a client c is reassigned to a different server, only the interaction paths involving c would change. Let A be the current client assignment. Then, the total length of all interaction paths involving a client c is given by

$$\begin{aligned} & \sum_{c_i \in C} \left(d(c, s_A(c)) + d(s_A(c), s_A(c_i)) + d(s_A(c_i), c_i) \right) \\ &= (|C| + 1)d(c, s_A(c)) + \sum_{c_i \neq c} d(s_A(c), s_A(c_i)) + \sum_{c_i \neq c} d(s_A(c_i), c_i), \end{aligned}$$

where $|C|$ is the number of clients. If client c is reassigned to a different server s , the total length of all interaction paths involving c becomes

$$(|C| + 1)d(c, s) + \sum_{c_i \neq c} d(s, s_A(c_i)) + \sum_{c_i \neq c} d(s_A(c_i), c_i).$$

Therefore, the reduction in interaction path length is

$$(|C| + 1)d(c, s_A(c)) + \sum_{c_i \neq c} d(s_A(c), s_A(c_i)) - \left((|C| + 1)d(c, s) + \sum_{c_i \neq c} d(s, s_A(c_i)) \right). \quad (\text{Eq. 4.3})$$

If we denote by $a(s)$ the number of clients that are assigned to server s in assignment A , $\sum_{c_i \neq c} d(s_A(c), s_A(c_i))$ can be rewritten as

$$\sum_{s_j \in S} a(s_j) d(s_A(c), s_j) - d(s_A(c), s_A(c)) = \sum_{s_j \in S} a(s_j) d(s_A(c), s_j).$$

Similarly, $\sum_{c_i \neq c} d(s, s_A(c_i))$ can be rewritten as

$$\sum_{s_j \in S} a(s_j) d(s, s_j) - d(s, s_A(c)).$$

Thus, the reduction (Eq. 4.3) can be rewritten as

$$(|C|+1)d(c, s_A(c)) + \sum_{s_j \in S} a(s_j) d(s_A(c), s_j) - \left((|C|+1)d(c, s) + \sum_{s_j \in S} a(s_j) d(s, s_j) - d(s, s_A(c)) \right). \quad (\text{Eq. 4.4})$$

To efficiently calculate the two summations in (Eq. 4.4), the values of $\sum_{s_j \in S} a(s_j) d(s, s_j)$ for all servers s can be stored in an array L and incrementally updated after each assignment modification. The pseudocode of *Modify Assignment* is presented in Algorithm 1. The calculation of array L for the initial assignment (lines 2 to 3) has a complexity of $O(|S|^2)$, where $|S|$ is the number of servers. The computational complexity of each assignment modification (lines 5 to 15) is $O(|S||C|)$. Thus, given an initial assignment, the total computational complexity of *Modify Assignment* is $O(|S|^2 + m|S||C|)$, where m is the number of assignment modifications performed before the algorithm terminates. Our experimental results in Section 4.3 would show that m is usually less than $|C|$.

In the absence of the triangle inequality, the example in Figure 4.5 also shows that *Modify Assignment* has unbounded approximation ratio (when it starts with c_1 and c_2 initially assigned to s_1 and s_2 respectively). If the network latency satisfies the triangle inequality, we can show that *Modify Assignment* has an approximation ratio of n irrespective of the initial client assignment, where n is the number of clients.

Theorem 4.5 *Irrespective of the initial client assignment, Modify Assignment has an approximation ratio of n for networks with the triangle inequality, where n is the number of clients.*

Proof Suppose there are n clients $c_1, c_2, \dots, c_n$ in the network. We denote by m_i the assigned server of client c_i in *Modify Assignment*, and denote by o_i the assigned server of c_i in an optimal assignment.

Algorithm 1: The *Modify Assignment* algorithm

```

1 for each server  $s_j$ , let  $a(s_j)$  be the number of clients assigned to  $s_j$  in the initial
  assignment;
2 foreach  $s \in S$  do
3    $L[s] \leftarrow \sum_{s_j \in S} a(s_j) \cdot d(s, s_j)$ ;
4 repeat
5    $\Delta^* \leftarrow -\infty$ ;
6   foreach  $s \in S$  do
7     foreach  $c \in C$  do
8        $\Delta \leftarrow (|C| + 1)d(c, s_A(c)) + L[s_A(c)] - ((|C| + 1)d(c, s) + L[s] - d(s, s_A(c)))$ ;
9       if  $\Delta > 0$  and  $\Delta > \Delta^*$  then
10         $\Delta^* \leftarrow \Delta$ ;
11         $c^* \leftarrow c$ ;  $s^* \leftarrow s$ ;
12   if  $\Delta^* > 0$  then
13     foreach  $s \in S$  do
14        $L[s] \leftarrow L[s] - d(s, s_A(c^*)) + d(s, s^*)$ ;
15       set  $s_A(c^*) = s^*$ ;
16 until  $\Delta^* \leq 0$  ;
    
```

When the *Modify Assignment* algorithm terminates, changing the assigned server of any client c_i to o_i cannot reduce the total length of all interaction paths involving c_i . Therefore, for each client c_i ,

$$\begin{aligned}
 & 2d(c_i, m_i) + \sum_{j \neq i} \left(d(c_i, m_i) + d(m_i, m_j) + d(m_j, c_j) \right) \\
 & \leq 2d(c_i, o_i) + \sum_{j \neq i} \left(d(c_i, o_i) + d(o_i, m_j) + d(m_j, c_j) \right),
 \end{aligned}$$

which implies that

$$(n + 1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j) \leq (n + 1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, m_j). \quad (\text{Eq. 4.5})$$

By the triangle inequality, we have

$$d(o_i, m_j) \leq d(o_i, c_i) + d(c_i, m_i) + d(m_i, m_j), \quad (\text{Eq. 4.6})$$

and

$$d(o_i, m_j) \leq d(o_i, o_j) + d(o_j, c_j) + d(c_j, m_j). \quad (\text{Eq. 4.7})$$

Combining inequalities (Eq. 4.5) and (Eq. 4.6), we obtain

$$\begin{aligned}
 & (n+1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j) \\
 & \leq (n+1)d(c_i, o_i) + \sum_{j \neq i} \left(d(o_i, c_i) + d(c_i, m_i) + d(m_i, m_j) \right) \\
 & = (n+1)d(c_i, o_i) + (n-1)d(o_i, c_i) + (n-1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j),
 \end{aligned}$$

which indicates that $d(c_i, m_i) \leq n \cdot d(c_i, o_i)$, and hence

$$n \cdot \sum_{i=1}^n d(c_i, m_i) \leq n^2 \cdot \sum_{i=1}^n d(c_i, o_i). \quad (\text{Eq. 4.8})$$

Combining inequalities (Eq. 4.5) and (Eq. 4.7), we have

$$\begin{aligned}
 & (n+1)d(c_i, m_i) + \sum_{j \neq i} d(m_i, m_j) \\
 & \leq (n+1)d(c_i, o_i) + \sum_{j \neq i} \left(d(o_i, o_j) + d(o_j, c_j) + d(c_j, m_j) \right).
 \end{aligned}$$

By adding up the above inequality for all clients c_i s, we obtain

$$\begin{aligned}
 & (n+1) \sum_{i=1}^n d(c_i, m_i) + \sum_{i=1}^n \sum_{j \neq i} d(m_i, m_j) \\
 & \leq (n+1) \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} \left(d(o_i, o_j) + d(o_j, c_j) + d(c_j, m_j) \right) \\
 & = (n+1) \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) + (n-1) \sum_{i=1}^n d(o_i, c_i) + (n-1) \sum_{i=1}^n d(c_i, m_i).
 \end{aligned}$$

It follows that

$$\sum_{i=1}^n d(c_i, m_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(m_i, m_j) \leq n \sum_{i=1}^n d(c_i, o_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j). \quad (\text{Eq. 4.9})$$

Finally, adding up (Eq. 4.8) and (Eq. 4.9), we have

$$\begin{aligned}
 & (n+1) \sum_{i=1}^n d(c_i, m_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(m_i, m_j) \\
 & \leq n(n+1) \sum_{i=1}^n d(c_i, o_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) \\
 & \leq n \left((n+1) \sum_{i=1}^n d(c_i, o_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) \right). \quad (\text{Eq. 4.10})
 \end{aligned}$$

Note that the left side of inequality (Eq. 4.10) is exactly the total length of all interaction paths in *Modify Assignment*, and the right side of inequality (Eq. 4.10) is exactly n times of the total length of all interaction paths in the optimal assignment. Therefore, *Modify Assignment* has an approximation ratio of n .

Hence, the theorem is proven. $\square$

Figure 4.6: The approximation ratio n of *Modify Assignment* and *Distributed-Modify Assignment* is tight.

The above approximation ratio n is tight if *Modify Assignment* starts with an arbitrary initial assignment. Figure 4.6 shows an example network containing n clients $c_1, c_2, \dots, c_n$ and two servers s_1, s_2 . The distance between each client and server s_1 is n , and the distance between each client and server s_2 is 1. Suppose that in the initial assignment, all clients are assigned to server s_1 . Then, the total length of all interaction paths involving any client c_i is $2n^2$. If client c_i is reassigned to server s_2 , the total length of all interaction paths involving c remains $2 + (2n + 2)(n - 1) = 2n^2$. Thus, *Modify Assignment* would terminate without reassigning any client. On the other hand, the optimal assignment is to assign all clients to server s_2 so that the interaction path length between any two clients is 2, which is $1/n$ of that in *Modify Assignment*. Therefore, the total interaction path length in *Modify Assignment* is n times of that in the optimal assignment.

The following analysis shows that, if *Modify Assignment* takes *Nearest-Server Assignment* as the initial assignment, it has an approximation ratio of 2.8. Our experimental

results in Section 4.3 would show that *Modify Assignment* normally improves interactivity over *Nearest-Server Assignment* significantly.

Suppose that there are n clients $c_1, c_2, \dots, c_n$ in the network. Denote the total interaction path lengths of *Modify Assignment* and an optimal assignment by $D_d(M)$ and $D_d(O)$ respectively. Denoted by $D_d(M^k)$ the total interaction path length of *Modify Assignment* after its k th assignment modification ($D_d(M^0)$ stands for that of the initial assignment, i.e., *Nearest-Server Assignment*). For each client c_i , let o_i be its assigned server in the optimal assignment, and s_i^k be the assigned server of c_i after the k th assignment modification of *Modify Assignment* (s_i^0 is the initial assigned server of client c_i , i.e., its nearest server). By definition, we have

$$\begin{aligned} D_d(M^k) &= \sum_{i=1}^n \sum_{j=i}^n \left(d(c_i, s_i^k) + d(s_i^k, s_j^k) + d(s_j^k, c_j) \right) \\ &= (n+1) \sum_{i=1}^n d(c_i, s_i^k) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i}^n d(s_i^k, s_j^k). \end{aligned}$$

and similarly,

$$D_d(O) = (n+1) \sum_{i=1}^n d(c_i, o_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i}^n d(o_i, o_j).$$

Lemma 4.1 *For networks with the triangle inequality, if Modify Assignment takes Nearest-Server Assignment as the initial assignment and performs at least r ($r \geq 0$) assignment modifications before it terminates, then*

$$\sum_{i=1}^n \Delta_i^r \geq D_d(M^r) + D_d(M^0) - \left(4 + \frac{8r}{n}\right) D_d(O),$$

where Δ_i^r is the change of the total interaction path length if client c_i is reassigned to its optimal server o_i after the r th assignment modification of *Modify Assignment*.

Proof For each $0 \leq k \leq r$, if we change the assigned server of client c_i from s_i^k to o_i after the k th assignment modification of *Modify Assignment*, according to (Eq. 4.3), Δ_i^k is given

by

$$\Delta_i^k = (n+1)d(c_i, s_i^k) + \sum_{j \neq i} d(s_i^k, s_j^k) - \left((n+1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, s_j^k) \right). \quad (\text{Eq. 4.11})$$

We first consider the case when $k = 0$. By the triangle inequality, we have

$$d(o_i, s_j^0) \leq d(o_i, o_j) + d(o_j, c_j) + d(c_j, s_j^0). \quad (\text{Eq. 4.12})$$

Combining the above two inequalities (Eq. 4.11) and (Eq. 4.12), we obtain

$$\begin{aligned} \Delta_i^0 &= (n+1)d(c_i, s_i^0) + \sum_{j \neq i} d(s_i^0, s_j^0) - \left((n+1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, s_j^0) \right) \\ &\geq (n+1)d(c_i, s_i^0) + \sum_{j \neq i} d(s_i^0, s_j^0) - \left((n+1)d(c_i, o_i) + \sum_{j \neq i} (d(o_i, o_j) + d(o_j, c_j) + d(c_j, s_j^0)) \right). \end{aligned}$$

By adding up the above inequality for all the clients $c_1, c_2, \dots, c_n$, we obtain

$$\begin{aligned} \sum_{i=1}^n \Delta_i^0 &\geq (n+1) \sum_{i=1}^n d(c_i, s_i^0) + \sum_{i=1}^n \sum_{j \neq i} d(s_i^0, s_j^0) \\ &\quad - \left((n+1) \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} (d(o_i, o_j) + d(o_j, c_j) + d(c_j, s_j^0)) \right) \\ &= (n+1) \sum_{i=1}^n d(c_i, s_i^0) + \sum_{i=1}^n \sum_{j \neq i} d(s_i^0, s_j^0) \\ &\quad - \left((n+1) \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) + (n-1) \sum_{i=1}^n d(o_i, c_i) + (n-1) \sum_{i=1}^n d(c_i, s_i^0) \right) \\ &= 2 \cdot \sum_{i=1}^n d(c_i, s_i^0) + \sum_{i=1}^n \sum_{j \neq i} d(s_i^0, s_j^0) - \left(2n \cdot \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) \right). \end{aligned}$$

Since s_i^0 is the nearest server of client c_i , we have

$$d(c_i, s_i^0) \leq d(c_i, o_i).$$

It follows that

$$\begin{aligned}
 \sum_{i=1}^n \Delta_i^0 &\geq 2 \cdot \sum_{i=1}^n d(c_i, s_i^0) + \sum_{i=1}^n \sum_{j \neq i} d(s_i^0, s_j^0) - \left(2n \cdot \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) \right) \\
 &\quad + 2n \cdot \sum_{i=1}^n d(c_i, s_i^0) - 2n \cdot \sum_{i=1}^n d(c_i, o_i) \\
 &= 2(n+1) \cdot \sum_{i=1}^n d(c_i, s_i^0) + \sum_{i=1}^n \sum_{j \neq i} d(s_i^0, s_j^0) - \left(4n \cdot \sum_{i=1}^n d(c_i, o_i) + \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) \right) \\
 &\geq 2 \left((n+1) \cdot \sum_{i=1}^n d(c_i, s_i^0) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(s_i^0, s_j^0) \right) \\
 &\quad - 4 \left((n+1) \cdot \sum_{i=1}^n d(c_i, o_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j) \right) \\
 &= 2D_d(M^0) - 4D_d(O). \tag{Eq. 4.13}
 \end{aligned}$$

Next, we consider the case when $k \geq 1$. Let c_{ι_k} be the client selected by *Modify Assignment* to perform its k th assignment modification. That is, in the k th assignment modification, the assigned server of client c_{ι_k} changes from $s_{\iota_k}^{k-1}$ to $s_{\iota_k}^k$. The assigned servers of all the clients other than c_{ι_k} remain unchanged in the k th assignment modification, i.e., for each $i \neq \iota_k$, we have

$$s_i^k = s_i^{k-1}. \tag{Eq. 4.14}$$

Thus, for each $i \neq \iota_k$, it follows from (Eq. 4.11) and (Eq. 4.14) that

$$\begin{aligned}
 \Delta_i^k &= (n+1)d(c_i, s_i^k) + \sum_{j \neq i} d(s_i^k, s_j^k) - \left((n+1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, s_j^k) \right) \\
 &= (n+1)d(c_i, s_i^{k-1}) + \sum_{j \neq i} d(s_i^{k-1}, s_j^{k-1}) - d(s_i^{k-1}, s_{\iota_k}^{k-1}) + d(s_i^{k-1}, s_{\iota_k}^k) \\
 &\quad - \left((n+1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, s_j^{k-1}) - d(o_i, s_{\iota_k}^{k-1}) + d(o_i, s_{\iota_k}^k) \right) \\
 &= (n+1)d(c_i, s_i^{k-1}) + \sum_{j \neq i} d(s_i^{k-1}, s_j^{k-1}) - \left((n+1)d(c_i, o_i) + \sum_{j \neq i} d(o_i, s_j^{k-1}) \right) \\
 &\quad - d(s_i^{k-1}, s_{\iota_k}^{k-1}) + d(s_i^{k-1}, s_{\iota_k}^k) + d(o_i, s_{\iota_k}^{k-1}) - d(o_i, s_{\iota_k}^k) \\
 &= \Delta_i^{k-1} - d(s_i^{k-1}, s_{\iota_k}^{k-1}) + d(s_i^{k-1}, s_{\iota_k}^k) + d(o_i, s_{\iota_k}^{k-1}) - d(o_i, s_{\iota_k}^k).
 \end{aligned}$$

Moreover,

$$\begin{aligned}
 \Delta_{\iota_k}^k &= (n+1)d(c_{\iota_k}, s_{\iota_k}^k) + \sum_{j \neq \iota_k} d(s_{\iota_k}^k, s_j^k) - \left((n+1)d(c_{\iota_k}, o_{\iota_k}) + \sum_{j \neq \iota_k} d(o_{\iota_k}, s_j^k) \right) \\
 &= (n+1)d(c_{\iota_k}, s_{\iota_k}^k) + \sum_{j \neq \iota_k} d(s_{\iota_k}^k, s_j^k) - \left((n+1)d(c_{\iota_k}, o_{\iota_k}) + \sum_{j \neq \iota_k} d(o_{\iota_k}, s_j^{k-1}) \right) \\
 &\quad + (n+1)d(c_{\iota_k}, s_{\iota_k}^{k-1}) + \sum_{j \neq \iota_k} d(s_{\iota_k}^{k-1}, s_j^{k-1}) - (n+1)d(c_{\iota_k}, s_{\iota_k}^{k-1}) - \sum_{j \neq \iota_k} d(s_{\iota_k}^{k-1}, s_j^{k-1}) \\
 &= (n+1)d(c_{\iota_k}, s_{\iota_k}^{k-1}) + \sum_{j \neq \iota_k} d(s_{\iota_k}^{k-1}, s_j^{k-1}) - \left((n+1)d(c_{\iota_k}, o_{\iota_k}) + \sum_{j \neq \iota_k} d(o_{\iota_k}, s_j^{k-1}) \right) \\
 &\quad - \left((n+1)d(c_{\iota_k}, s_{\iota_k}^{k-1}) + \sum_{j \neq \iota_k} d(s_{\iota_k}^{k-1}, s_j^{k-1}) - (n+1)d(c_{\iota_k}, s_{\iota_k}^k) - \sum_{j \neq \iota_k} d(s_{\iota_k}^k, s_j^k) \right) \\
 &= \Delta_{\iota_k}^{k-1} - \left((n+1)d(c_{\iota_k}, s_{\iota_k}^{k-1}) + \sum_{j \neq \iota_k} d(s_{\iota_k}^{k-1}, s_j^{k-1}) - (n+1)d(c_{\iota_k}, s_{\iota_k}^k) - \sum_{j \neq \iota_k} d(s_{\iota_k}^k, s_j^k) \right).
 \end{aligned}$$

Note that the change of the total interaction path length in the k th assignment modification,

i.e., $D_d(M^{k-1}) - D_d(M^k)$, is given by

$$(n+1)d(c_{\iota_k}, s_{\iota_k}^{k-1}) + \sum_{j \neq \iota_k} d(s_{\iota_k}^{k-1}, s_j^{k-1}) - \left((n+1)d(c_{\iota_k}, s_{\iota_k}^k) + \sum_{j \neq \iota_k} d(s_{\iota_k}^k, s_j^k) \right).$$

Therefore,

$$\Delta_{\iota_k}^k = \Delta_{\iota_k}^{k-1} + D_d(M^k) - D_d(M^{k-1}).$$

By summing up Δ_i^k for all the clients $c_1, c_2, \dots, c_n$, we have

$$\begin{aligned}
 \sum_{i=1}^n \Delta_i^k &= \sum_{i=1}^n \Delta_i^{k-1} + D_d(M^k) - D_d(M^{k-1}) \\
 &\quad + \sum_{i \neq \iota_k} \left(-d(s_i^{k-1}, s_{\iota_k}^{k-1}) + d(s_i^{k-1}, s_{\iota_k}^k) + d(o_i, s_{\iota_k}^{k-1}) - d(o_i, s_{\iota_k}^k) \right).
 \end{aligned}$$

By the triangle inequality, it follows that

$$\begin{aligned}
 \sum_{i=1}^n \Delta_i^k &\geq \sum_{i=1}^n \Delta_i^{k-1} + D_d(M^k) - D_d(M^{k-1}) \\
 &\quad + \sum_{i \neq \iota_k} \left(-d(s_i^{k-1}, c_i) - d(c_i, s_{\iota_k}^{k-1}) + d(c_i, s_{\iota_k}^k) - d(c_i, s_i^{k-1}) \right. \\
 &\quad \left. + d(c_i, s_{\iota_k}^{k-1}) - d(c_i, o_i) - d(o_i, c_i) - d(c_i, s_{\iota_k}^k) \right) \\
 &= \sum_{i=1}^n \Delta_i^{k-1} + D_d(M^k) - D_d(M^{k-1}) - 2 \cdot \sum_{i \neq \iota_k} \left(d(c_i, s_i^{k-1}) + d(c_i, o_i) \right) \\
 &\geq \sum_{i=1}^n \Delta_i^{k-1} + D_d(M^k) - D_d(M^{k-1}) - 2 \cdot \sum_{i=1}^n \left(d(c_i, s_i^{k-1}) + d(c_i, o_i) \right). \quad (\text{Eq. 4.15})
 \end{aligned}$$

Note that

$$D_d(M^{k-1}) = (n+1) \sum_{i=1}^n d(c_i, s_i^{k-1}) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(s_i^{k-1}, s_j^{k-1})$$

cannot be greater than $3D_d(O)$ since the assignment modifications only reduce the total interaction path length starting from *Nearest-Server Assignment* which has an approximation ratio of 3. Thus, it follows that

$$\sum_{i=1}^n d(c_i, s_i^{k-1}) \leq \frac{3D_d(O)}{n+1} \leq \frac{3D_d(O)}{n}.$$

Similarly, since

$$D_d(O) = (n+1) \sum_{i=1}^n d(c_i, o_i) + \frac{1}{2} \sum_{i=1}^n \sum_{j \neq i} d(o_i, o_j),$$

we have

$$\sum_{i=1}^n d(c_i, o_i) \leq \frac{D_d(O)}{n+1} \leq \frac{D_d(O)}{n}.$$

Thus, inequality (Eq. 4.15) becomes

$$\sum_{i=1}^n \Delta_i^k \geq \sum_{i=1}^n \Delta_i^{k-1} + D_d(M^k) - D_d(M^{k-1}) - \frac{8D_d(O)}{n}.$$

By combining the above inequalities for all $k = 1, 2, \dots, r$ and inequality (Eq. 4.13), we

have

$$\begin{aligned}
 \sum_{i=1}^n \Delta_i^r &\geq \sum_{i=1}^n \Delta_i^0 + \sum_{k=1}^r \left(D_d(M^k) - D_d(M^{k-1}) \right) - \frac{8r}{n} D_d(O) \\
 &\geq 2D_d(M^0) - 4D_d(O) + D_d(M^r) - D_d(M^0) - \frac{8r}{n} D_d(O) \\
 &= D_d(M^r) + D_d(M^0) - \left(4 + \frac{8r}{n} \right) D_d(O).
 \end{aligned}$$

Hence, the lemma is proven. $\square$

Theorem 4.6 *Taking Nearest-Server Assignment as the initial assignment, Modify Assignment has an approximation ratio of 2.8 for networks with the triangle inequality.*

Proof We start by studying the total interaction path length produced by *Modify Assignment* assuming that it terminates after a given number of t ($t \geq 0$) assignment modifications.

First, since the total interaction path length cannot be reduced by changing the assigned server of any client after the t th assignment modification, we have $\Delta_i^t \leq 0$ for any client c_i .

According to Lemma 4.1, we have

$$0 \geq \sum_{i=1}^n \Delta_i^t \geq D_d(M^t) + D_d(M^0) - \left(4 + \frac{8t}{n} \right) D_d(O),$$

which implies that

$$D_d(M) = D_d(M^t) \leq \left(4 + \frac{8t}{n} \right) D_d(O) - D_d(M^0).$$

If the initial assignment M^0 satisfies $D_d(M^0) \geq \left(2 + \frac{4t}{n} \right) D_d(O)$, we have $D_d(M) \leq \left(4 + \frac{8t}{n} \right) D_d(O) - \left(2 + \frac{4t}{n} \right) D_d(O) = \left(2 + \frac{4t}{n} \right) D_d(O)$. Otherwise, if $D_d(M^0) \leq \left(2 + \frac{4t}{n} \right) D_d(O)$, we have $D_d(M) \leq D_d(M^0) \leq \left(2 + \frac{4t}{n} \right) D_d(O)$. Therefore, in either case, the total interaction path length of *Modify Assignment* satisfies

$$D_d(M) \leq \left(2 + \frac{4t}{n} \right) D_d(O). \quad (\text{Eq. 4.16})$$

Second, we consider the r th ($1 \leq r \leq t$) assignment modification performed by *Modify Assignment*. Since *Modify Assignment* always selects the reassignment option that produces

the maximum reduction in the total interaction path length, the reduction must not be smaller than the average reduction of the reassignments that reassign client c_i to server o_i , i.e., $D_d(M^{r-1}) - D_d(M^r) \geq \frac{1}{n} \sum_{i=1}^n \Delta_i^{r-1}$. Therefore, according to Lemma 4.1, we have

$$D_d(M^{r-1}) - D_d(M^r) \geq \frac{1}{n} \left(D_d(M^{r-1}) + D_d(M^0) - \left(4 + \frac{8(r-1)}{n} \right) D_d(O) \right),$$

which can be rewritten as

$$D_d(M^r) \leq \left(1 - \frac{1}{n} \right) D_d(M^{r-1}) - \frac{1}{n} D_d(M^0) + \left(\frac{4}{n} + \frac{8(r-1)}{n^2} \right) D_d(O).$$

By integrating the above inequalities from the first to the r th assignment modifications performed, we have

$$\begin{aligned} D_d(M^r) &\leq \left(1 - \frac{1}{n} \right)^r D_d(M^0) - \left(\frac{1}{n} + \left(1 - \frac{1}{n} \right) \frac{1}{n} + \cdots + \left(1 - \frac{1}{n} \right)^{r-1} \frac{1}{n} \right) D_d(M^0) \\ &\quad + \left(\left(\frac{4}{n} + \frac{8(r-1)}{n^2} \right) + \left(1 - \frac{1}{n} \right) \left(\frac{4}{n} + \frac{8(r-2)}{n^2} \right) + \cdots \right. \\ &\quad \left. + \left(1 - \frac{1}{n} \right)^{r-2} \left(\frac{4}{n} + \frac{8}{n^2} \right) + \left(1 - \frac{1}{n} \right)^{r-1} \frac{4}{n} \right) D_d(O) \\ &= \left(2 \left(1 - \frac{1}{n} \right)^r - 1 \right) D_d(M^0) + \left(4 - 4 \left(1 - \frac{1}{n} \right)^r + \frac{8r}{n} - 8 + 8 \left(1 - \frac{1}{n} \right)^r \right) D_d(O). \end{aligned}$$

Since $D_d(M^0) \leq 3D_d(O)$ (Theorem 4.4), it follows that

$$D_d(M) \leq D_d(M^r) \leq \left(\frac{8r}{n} - 7 + 10 \left(1 - \frac{1}{n} \right)^r \right) D_d(O).$$

Note that the above inequality holds for all $r = 1, 2, \dots, t$. Thus, for any $t \geq 1$, we have

$$D_d(M) \leq \min_{1 \leq r \leq t} \left\{ \frac{8r}{n} - 7 + 10 \left(1 - \frac{1}{n} \right)^r \right\} D_d(O). \quad (\text{Eq. 4.17})$$

If $t \geq 1$, combining (Eq. 4.16) and (Eq. 4.17), we obtain

$$D_d(M) \leq \min \left\{ 2 + \frac{4t}{n}, \min_{1 \leq r \leq t} \left\{ \frac{8r}{n} - 7 + 10 \left(1 - \frac{1}{n} \right)^r \right\} \right\} D_d(O).$$

Considering all the possible values of t , it follows that

$$D_d(M) \leq \max_{t \geq 1} \min \left\{ 2 + \frac{4t}{n}, \min_{1 \leq r \leq t} \left\{ \frac{8r}{n} - 7 + 10 \left(1 - \frac{1}{n} \right)^r \right\} \right\} D_d(O).$$

Let $f(t) = 2 + \frac{4t}{n}$ and $g(t) = \min_{1 \leq r \leq t} \left\{ \frac{8r}{n} - 7 + 10\left(1 - \frac{1}{n}\right)^r \right\}$. It is obvious that $f(t)$ monotonically increases with t . On the other hand, since

$$\frac{d^2\left(\frac{8r}{n} - 7 + 10\left(1 - \frac{1}{n}\right)^r\right)}{dr^2} = 10\left(1 - \frac{1}{n}\right)^r \ln^2\left(1 - \frac{1}{n}\right) + \frac{10\left(1 - \frac{1}{n}\right)^r}{1 - \frac{1}{n}} > 0,$$

$\frac{8r}{n} - 7 + 10\left(1 - \frac{1}{n}\right)^r$ is concave with respect to r and reaches its minimum when $\frac{d\left(\frac{8r}{n} - 7 + 10\left(1 - \frac{1}{n}\right)^r\right)}{dr} = 0$, i.e., when $r = \frac{\ln \frac{8}{10n \ln \frac{n}{n-1}}}{\ln \frac{n-1}{n}}$. Thus, the value of $g(t)$ monotonically decreases with t and reaches its minimum when $t \geq t^* = \frac{\ln \frac{8}{10n \ln \frac{n}{n-1}}}{\ln \frac{n-1}{n}}$ as illustrated in Figure 4.7. That is,

$$g(t) = \begin{cases} \frac{8t}{n} - 7 + 10\left(1 - \frac{1}{n}\right)^t & \text{if } t < t^*, \\ \frac{8t^*}{n} - 7 + 10\left(1 - \frac{1}{n}\right)^{t^*} & \text{if } t \geq t^*. \end{cases}$$

Note that

$$\begin{aligned} f(t^*) - g(t^*) &= 2 + \frac{4t^*}{n} - \frac{8t^*}{n} + 7 - 10\left(1 - \frac{1}{n}\right)^{t^*} \\ &= 9 - \frac{4t^*}{n} - 10\left(1 - \frac{1}{n}\right)^{t^*} \\ &= 9 + \frac{4 \ln \frac{8}{10n \ln \frac{n}{n-1}}}{n \ln \frac{n-1}{n}} - \frac{8}{n \ln \frac{n-1}{n}}. \end{aligned}$$

Since $\frac{d(f(t^*) - g(t^*))}{d\left(n \ln \frac{n}{n-1}\right)} = 9 - \frac{4}{n \ln \frac{n}{n-1}}$ and $n \ln \frac{n}{n-1} > 1$, we have $\frac{d(f(t^*) - g(t^*))}{d\left(n \ln \frac{n}{n-1}\right)} > 0$. Thus, $f(t^*) - g(t^*)$ monotonically increases with $n \ln \frac{n}{n-1}$. As a result, $f(t^*) - g(t^*) > 9 + 4 \ln \frac{8}{10} - 8 > 0$.

Hence, $f(t^*) > g(t^*)$. Therefore, as illustrated in Figure 4.7, $\max_{t \geq 1} \min\{f(t), g(t)\}$ must be achieved at the intersection of $f(t)$ and $g(t)$, and given any value t_0 , $\max_{t \geq 1} \min\{f(t), g(t)\} \leq \max\{f(t_0), g(t_0)\}$. By making use of $\left(1 - \frac{1}{n}\right)^t \approx 1 - \frac{t}{n}$, we can obtain an approximation solution $t = \frac{n}{6}$ to the equation $f(t) = g(t)$. Thus, it follows that

$$\begin{aligned} \max_{t \geq 1} \min\{f(t), g(t)\} &\leq \max\left\{f\left(\frac{n}{6}\right), g\left(\frac{n}{6}\right)\right\} \\ &= \max\left\{\frac{8}{3}, 10\left(1 - \frac{1}{n}\right)^{\frac{n}{6}} - \frac{17}{3}\right\} \\ &\leq 10 \cdot \left(\frac{1}{e}\right)^{\frac{1}{6}} - \frac{17}{3} \\ &< 2.8, \end{aligned}$$

which implies that for any $t \geq 1$,

$$D_d(M) < 2.8D_d(O).$$

If $t = 0$, according to (Eq. 4.16), we have $D_d(M) \leq 2D_d(O)$.

Hence, the theorem is proven. $\square$

Figure 4.7: Curves of $f(t)$ and $g(t)$ for $n = 1000$.

4.2.3 Distributed-Modify Assignment

Distributed-Modify Assignment is a distributed version of *Modify Assignment*, in which assignment modifications are performed in a distributed manner. Starting from an initial assignment, each client individually changes its assigned server to the server that decreases the total interaction path length most. The algorithm terminates if no client can reduce the total interaction path length by changing its assigned server.

Similar to *Modify Assignment*, a client c only needs to consider the interaction paths involving itself in performing an assignment modification. As derived in Section 4.2.2, the total length of these interaction paths is given by $(|C| + 1)d(c, s) + \sum_{s_j \in S} a(s_j)d(s, s_j) -$

$d(s, s_A(c)) + \sum_{c_i \neq c} d(s_A(c_i), c_i)$, if c changes its assigned server to s . Note that the last summation is independent of s and can be omitted for comparison purposes. Thus, this expression becomes

$$(|C| + 1)d(c, s) + \sum_{s_j \in S} a(s_j)d(s, s_j) - d(s, s_A(c)). \quad (\text{Eq. 4.18})$$

The calculation needs the information of the latency $d(c, s)$ between c and s , the latencies between s and other servers, and the number of clients assigned to each server s_j (i.e., $a(s_j)$). To this end, each server maintains the latencies from itself to the other servers, and one master server keeps track of how many clients are assigned to each server. Note that the last two terms of (Eq. 4.18), which we shall denote by $H(s) = \sum_{s_j \in S} a(s_j)d(s, s_j) - d(s, s_A(c))$, are independent of the client-to-server latency. To perform an assignment modification, client c 's server $s_A(c)$ queries the master server and calculates $H(s)$ for all the other servers $s \neq s_A(c)$. If the current value of (Eq. 4.18), i.e., $(|C|+1)d(c, s_A(c)) + H(s_A(c))$, is not greater than $H(s)$, changing c 's server to s cannot reduce the interaction path length. Thus, $s_A(c)$ informs c about the list of servers whose $H(s)$ is less than $(|C| + 1)d(c, s_A(c)) + H(s_A(c))$. Then, c contacts each server s in the list to measure the latency to the server and calculates the value of (Eq. 4.18) for this server. Based on the calculation results, c decides locally whether and how to change its assigned server. The computational complexity and message complexity of an assignment modification are $O(|S|^2)$ and $O(|S|)$ respectively.

Figure 4.8: An example in which changing the assigned servers of two clients simultaneously increases the total interaction path length.

One issue with *Distributed-Modify Assignment* is that, if two or more clients change their assigned servers simultaneously, the total interaction path length is not guaranteed to

decrease because the calculation of each client is based on the assumption that the assigned servers of other clients remain unchanged. Figure 4.8 gives an example, where the network contains clients c_1 , c_2 and servers s_1 , s_2 , s_3 , s_4 , and the number on each link represents the length of the link. Suppose the initial assignment assigns c_1 to s_1 and c_2 to s_2 . The lengths of interaction paths between client pairs (c_1, c_2) , (c_1, c_1) and (c_2, c_2) are 30, 20 and 20 respectively. So, the total path length is 70. When client c_1 tries to change its assigned server, it would choose s_3 since the length of the interaction path between c_1 and c_2 would be reduced to 25 while the lengths of interaction paths between client pairs (c_1, c_1) and (c_2, c_2) stay the same, assuming that c_2 keeps its assigned server unchanged. Similarly, client c_2 would choose to change its assigned server to s_4 . However, if the two clients both change their assigned servers at the same time, the path length between c_1 and c_2 would be increased to 40 which is even longer than that in the initial assignment. This issue can be resolved by a concurrency control mechanism to ensure that assignment modification proceeds one client at a time. To do so, a token can be maintained by the servers and circulated among clients. Only the client obtaining the token is allowed to perform an assignment modification. After the assignment modification, the client returns the token to its server which passes the token onto another client.

If the network latency satisfies the triangle inequality, *Distributed-Modify Assignment* has the same tight approximation ratio n as *Modify Assignment* irrespective of the initial client assignment, where n is the number of clients. The proof of Theorem 4.5 and the example of Figure 4.6 also apply to *Distributed-Modify Assignment*. In the absence of the triangle inequality, *Distributed-Modify Assignment* has unbounded approximation ratio. The example of Figure 4.5 is also applicable to *Distributed-Modify Assignment*.

4.2.4 Dealing with Server Capacity Constraints

So far, our proposed assignment algorithms have not assumed any capacity limitation at the servers. These “uncapacitated” algorithms are suitable for the scenario where each

Figure 4.9: The approximation ratio of the “capacitated” *Nearest-Server Assignment* is larger than 3.

server site has abundant server resources or server resources can be added to these sites as needed [54]. However, if the server capacity at each site is limited, assigning more clients to a server than its capacity may result in significant increase in the processing delay at the server, damaging the interactivity of the DIA [57]. Simple steps can be taken in our proposed algorithms to ensure that the capacity of each server is not exceeded. In *Nearest-Server Assignment*, if the nearest server of a client is saturated, the client turns to the next nearest server and keeps doing so until it finds a server that can accommodate more clients. For *Modify Assignment* and *Distributed-Modify Assignment*, only the servers not yet saturated are considered when performing assignment modifications. We evaluate both “uncapacitated” and “capacitated” assignment algorithms in the next section.

The approximation ratios of the “uncapacitated” algorithms do not apply directly to the “capacitated” algorithms. Figure 4.9 gives an example when the “capacitated” *Nearest-Server Assignment* is more than three times worse than the optimal assignment. Each server in this network can accommodate one client only. Suppose that in *Nearest-Server Assignment*, client c_1 is first assigned to server s_2 . Then, each remaining client c_i ($i = 2, 3, 4$) can only be assigned to server s_{i+1} . The total interaction path length under this assignment is 1190. The optimal assignment should assign each c_i ($i = 1, 2, 3, 4$) to s_i , which produces a total interaction path length of 322. Thus, the ratio between the results of the two assignments is 3.7.

4.3 Experimental Evaluation

4.3.1 Experimental Setup

To evaluate the proposed assignment algorithms, we have conducted extensive simulation experiments by making use of the Meridian data set [1]. This is a real network latency data set that contains pair-wise latency measurements between 2,500 nodes in the Internet using the King technique [38]. The measurements for some node pairs are not available in the data set. On discarding the nodes involved in unavailable measurements, the network simulated in our experiments is represented by a complete pair-wise latency matrix between 1,796 nodes. A client is assumed to be located at each node and a certain number of servers are assumed to be placed at selected nodes in the network. The proposed client assignment algorithms are evaluated with different parameter settings of server number and server capacity. In our experiments, we choose *Nearest-Server Assignment* as the initial assignment for both *Modify Assignment* and *Distributed-Modify Assignment*. In the default setting, no limitation is assumed on the server capacity.

We simulate three types of server placements: random, k -median and k -center. As discussed in Section 2.3, the k -median and k -center problems are widely used to model server placement in the Internet [26, 63]. Both problems are NP-hard [34]. We adopt a greedy k -median heuristic [63] and a greedy k -center heuristic [26] for placing servers in our experiments. For random server placement, under each parameter setting, we perform 1,000 simulation runs using 1,000 different sets of servers placed at random. Unless stated otherwise, we plot for comparison the average performance of these simulation runs together with the 90th and 10th percentile results.

To quantify the relative performance difference, the interactivity performance of different algorithms is normalized with respect to a common theoretical lower bound, which is derived as follows. For any client assignment A , the interaction path between two clients u, v has a

length of

$$\begin{aligned} & d(u, s_A(u)) + d(s_A(u), s_A(v)) + d(s_A(v), v) \\ & \geq \min_{s_i, s_j \in S} \{d(u, s_i) + d(s_i, s_j) + d(s_j, v)\}. \end{aligned}$$

By adding up the above inequalities for all client pairs, we get the following lower bound on the total interaction path length:

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

Note that this lower bound is a super bound that may not be achievable by any client assignment. This is because the above bound calculation does not enforce the constraint that each client is assigned to only one server through which it interacts with all the other clients. The total length of interaction paths produced by each algorithm normalized by the above bound shall be called the *normalized interactivity*.

We also include in the comparison an existing heuristic *MAXFLO* proposed in [18] for a similar problem called the fixed-hub single allocation problem. We adapt this heuristic to our client assignment problem. The first step of the heuristic is similar to the above calculation of the lower bound, in which we find the assigned servers for each pair of clients that produce the shortest interaction path between the two clients. In the second step, each client is assigned to the server to which it is assigned for the most times in the first step. We compare *MAXFLO* with our “uncapacitated” algorithms.

4.3.2 Number of Assignment Modifications

We first investigate the number of assignment modifications performed by *Modify Assignment* and *Distributed-Modify Assignment*. Figure 4.10 shows the distribution of the numbers of assignment modifications required for *Modify Assignment* to terminate under 1,000 different sets of 80 randomly placed servers. The number of modifications is normally less than the number of clients (i.e., each client performs less than one modification on average).

Figure 4.10: Distribution of the number of assignment modifications performed by *Modify Assignment* for networks with 80 randomly placed servers.

Figure 4.11: Performance of *Distributed-Modify Assignment* after different numbers of stages of assignment modifications for networks with 80 servers.

The numbers of assignment modifications under the k -median and k -center placements are measured to be 1,620 and 1,289 respectively, which are also less than the number of clients. Similar observations are made for other server numbers as well.

Unlike *Modify Assignment* which performs assignment modifications based on global knowledge, *Distributed-Modify Assignment* makes modification decisions locally at individual clients. Therefore, *Distributed-Modify Assignment* generally needs more assignment modifications than *Modify Assignment*. We count an attempt made by a client to change its assigned server as one assignment modification, even if the client decides not to change

its assigned server after calculation. n successive assignment modifications are referred to as *one stage* if there are n clients. We study the convergence speed of *Distributed-Modify Assignment* by monitoring its interactivity performance after each stage of assignment modifications. Two schemes for circulating the token among clients are tested: random and round-robin. Random circulation passes the token to a randomly selected client after each assignment modification, whereas round-robin circulation passes the token to each client exactly once in a stage of assignment modifications. Figure 4.11 presents the results for 80 servers under different server placements. As can be seen, the normalized interactivity is continuously improved with increasing number of stages and the algorithm converges quickly in just a few stages. Random and round-robin circulations achieve more than 90% and 99% of the improvements over the initial assignment respectively within three stages (i.e., after each client has performed three assignment modifications on average). Similar trends are also observed for other server numbers. Thus, in the following experiments, we shall execute the *Distributed-Modify Assignment* algorithm for only three stages of assignment modifications for both token passing schemes.

4.3.3 Performance Comparison of Different Algorithms

Now, we compare the performance of different client assignment algorithms. Figure 4.12 shows the normalized interactivity produced by the algorithms for different server numbers and server placements. It can be seen that *Modify Assignment* produces the best results, and *Distributed-Modify Assignment* performs very close to its centralized counterpart. The interactivities resulting from *Modify Assignment* and *Distributed-Modify Assignment* are normally within 40% to 50% of the lower bound, which means that they are within at most the same percentages of the optimum. Round-robin circulation can improve the performance of *Distributed-Modify Assignment* over random circulation, but the improvement is limited (within 5%). Note that additional information needs to be maintained by the server to

Table 4.1: Computation times for networks with 80 servers

Client assignment algorithm	CPU time (second)		
	Random	k -median	k -center
<i>Nearest-Server</i>	0.16	0.17	0.16
<i>Modify</i>	18.75	17.27	13.58
<i>Distributed-Modify</i> (Random)	2.13	2.14	2.18
<i>Distributed-Modify</i> (Round-robin)	3.52	3.54	3.47
<i>MAXFLO</i>	27.40	36.43	24.07

ensure that each client obtains the token exactly once in a stage of assignment modifications. Thus, we shall use random circulation for passing the token in the following experiments. *Nearest-Server Assignment* produces the worst interactivity among all the algorithms. This implies that reducing the client-to-server latency alone is not very effective in improving interactivity. *MAXFLO* outperforms *Nearest-Server Assignment*, but its performance is still far worse than our modification-based algorithms.

Table 4.1 reports the computation times of the algorithms under 80 servers of various placements in our simulation performed on a PC with a Pentium Xeon 3.2 GHz CPU and 6GB RAM. It can be seen that *Distributed-Modify Assignment* has much shorter computation time than *Modify Assignment* and *MAXFLO*.

4.3.4 Impact of Server Capacity

We also evaluate the “capacitated” version of the assignment algorithms. We refer to the maximum number of clients that can be assigned to each server as the server capacity. Figure 4.13 shows the normalized interactivity as a function of the server capacity when there are 80 servers placed in the network. Note that the theoretical lower bound does not change with server capacity as it assumes unlimited server capacity. As can be seen, the interactivity resulting from all algorithms generally becomes worse with decreasing server capacity. This is because lower server capacities make the algorithms unable to assign the clients to the most desirable servers. In general, the relative performance of all algorithms

(a) Random placement

 (b) k -median placement

 (c) k -center placement

Figure 4.12: Performance of client assignment algorithms for different numbers of servers.

(a) Random placement

 (b) k -median placement

 (c) k -center placement

Figure 4.13: Performance of client assignment algorithms for different server capacities.

remains unchanged over different server capacities. Only when the server capacity is severely limited do all algorithms produce similar interactivity, which is much worse than that when servers have abundant capacities. Similar results are observed for other server numbers.

4.3.5 Dynamic Scenarios

The above experiments have shown that *Modify Assignment* results in the best interactivity among all algorithms in static scenarios where the participating clients and network latencies are fixed. However, *Modify Assignment* is a centralized algorithm that requires global knowledge about the network. On the other hand, *Distributed-Modify Assignment*, which performs close to *Modify Assignment*, is a distributed algorithm that is more suitable for incrementally adapting the client assignment to dynamics in client participation and network latencies. In this section, we evaluate the adaptivity of *Distributed-Modify Assignment* in dynamic scenarios. For comparison purposes, we also include *Nearest-Server Assignment* in the experiments.

4.3.5.1 Adaptivity to Client Joining/Leaving

Figure 4.14: The number of participating clients over the event sequence.

We first consider the scenario in which the set of participating clients changes over time due to joins and leaves. We simulate it by a sequence of randomly generated client join and

leave events. To model clients with different join and leave patterns, each client is assigned two random weights w_j and w_l ($0 < w_j, w_l < 1$). If a client is currently participating, the probability of generating a leave event for the client is proportional to its w_j ; if a client is not participating, the probability of generating a join event for the client is proportional to its w_l . As a result, clients with higher w_j and w_l weights would join and leave the system more frequently than clients with lower w_j and w_l weights. Clients with high w_j but low w_l tend to participate in the system for longer time periods than clients with low w_j but high w_l . The simulation run starts with no client participating in the system and executes a sequence of 10,000 join and leave events. Figure 4.14 shows the number of clients participating in the system over the event sequence. As can be seen, after a warm-up period of about 5,000 events, the number of participating clients remains stable at about 900, which is about half the total number of clients in the network because the weights of all clients are randomly selected.

To adapt the client assignment to dynamic client participation, *Nearest-Server Assignment* simply assigns all joining clients to their nearest servers. For *Distributed-Modify Assignment*, a certain number of assignment modifications are performed after each join and leave event to refine the assignment. In the case of a join event, the modifications start with the joining client. Note that all assignment modifications are made based on the existing client assignment. Figure 4.15 shows the interactivity performance of different algorithms when there are 80 servers placed in the network. The results presented for random server placement are those for one sample set of randomly placed servers. Other sets of servers tested show similar performance trends. As can be seen, while aggressively performing three stages of assignment modifications at every event produces the best interactivity among all algorithms, performing as few as just one assignment modification at every event results in almost the same performance. This implies that *Distributed-Modify Assignment* is able to

(a) One sample random placement

 (b) k -median placement

 (c) k -center placement

Figure 4.15: Performance of client assignment algorithms with dynamic client participation.

adapt the client assignment quickly to dynamic client participation. Performing only a few assignment modifications per event is sufficient to maintain decent interactivity performance.

We also study the communication overhead of *Distributed-Modify Assignment* by counting the number of servers that are contacted by the client in each assignment modification. On average, the numbers of servers contacted in an assignment modification are 38, 26 and 25.5 for random, k -median and k -center server placements respectively, which are less than 1/3 to half of all the servers in the network.

4.3.5.2 Adaptivity to Dynamics in Network Latency

The network latency between nodes may change over time due to dynamics in network conditions. In this section, we evaluate the adaptivity of the algorithms to the variance in network latency. Since the Meridian data set does not contain latency data at different times, we use the PlanetLab-All-Pairs-Pings data set [2] which includes periodic pair-wise latency measurements between PlanetLab nodes. The measurements were performed approximately once every 15 minutes. We use the data collected over a one day period (on May 9, 2005) that contains 95 measurements. There are 188 PlanetLab nodes whose pair-wise latency measurements are consistently available in all of these 95 measurements. Thus, we simulate a network consisting of these 188 nodes. Similar to the previous experiments, a client is assumed to be located at each node and a total number of 15 servers are assumed to be placed at selected nodes in the network.

To deal with dynamic network latency, *Nearest-Server Assignment* simply lets each client recalculate its nearest server after every measurement. For *Distributed-Modify Assignment*, three stages of assignment modifications are performed after every measurement to refine the existing client assignment.

Figure 4.16 shows the interactivity performance of the two algorithms over the one day period. Here, the k -median and k -center server placements are computed based on the

(a) One sample random placement

 (b) k -median placement at initial latency measurement

 (c) k -center placement at initial latency measurement

Figure 4.16: Performance of client assignment algorithms with dynamic network latency.

Table 4.2: Number of clients reassigned after three stages of assignment modifications

Server placement	Average number of clients reassigned
Random	10
k -median	9.98
k -center	12.93

latency data of the first measurement. This models the scenario in which the network latency information used for the deployment of servers is predicted by the first measurement. The theoretical lower bound on the total length of interaction paths is recalculated after each measurement for computing the normalized interactivity. It can be seen from Figure 4.16 that *Distributed-Modify Assignment* almost always outperforms *Nearest-Server Assignment*. The normalized interactivity of *Nearest-Server Assignment* fluctuates widely. This is because *Nearest-Server Assignment* considers the client-to-server latencies only. Thus, its relative performance would become worse when the interserver latencies between the nearest servers of the clients increase. In contrast, *Distributed-Modify Assignment* can adjust to changes in both the client-to-server latencies and interserver latencies. As a result, its normalized interactivity remains quite stable over time, implying that it adapts well to the variance in network latency. In *Distributed-Modify Assignment*, the average numbers of servers contacted by the client in each assignment modification are 7.89, 4.98 and 2.97 for random, k -median and k -center server placements respectively, which are typically less than half of the total server number, similar to the results in the previous scenario. Note that the client that performs an assignment modification is not necessarily reassigned. As shown in Table 4.2, only less than 7% clients change their assigned servers after each measurement although three stages of assignment modifications are performed.

4.4 Optimal Assignment for Tree Topologies

Overlay tree topologies are widely used to facilitate large-scale group communication in distributed applications [41]. For example, a shared spanning tree topology can be built

for efficiently supporting multicast communication among distributed servers and from the servers to their clients [47]. While the client assignment problem for discrete DIAs is NP-complete for general topologies, there do exist polynomial time optimal solutions for tree topologies. In this section, we develop a dynamic programming algorithm to compute the optimal client assignment in tree topologies. For simplicity, we first assume that there is a client located at each node in the network. We extend the solution to the case where clients are located at only a subset of nodes in Section 4.4.3.

4.4.1 Generalized Problem Definitions

In a tree topology, each internal node u and its descendant nodes form a subtree rooted at u . As shown in Figure 4.17, we denote the subtree rooted at node u by T_u and denote the set of links incident upon the nodes in subtree T_u by E_u . Note that E_u includes all links whose two ends are both located inside T_u as well as the link connecting u to its parent $p(u)$ if u is not the root of the entire tree. We denote by $\mathcal{C}(E_u)$ the contribution of the links E_u to the total length of all interaction paths between clients. It is obvious that $\mathcal{C}(E_u)$ is affected only by the assignment of the clients inside T_u and how many clients outside T_u are assigned to servers inside T_u . Thus, to develop a dynamic programming algorithm, we consider a generalized problem of how to assign the clients inside T_u to minimize the contribution $\mathcal{C}(E_u)$ given that a certain number of clients outside T_u are assigned to servers

Figure 4.17: The subtree T_u rooted at node u and the link set E_u .

inside T_u .

Note that the path between any two nodes in a tree is unique and is the shortest path connecting these two nodes. Thus, both Theorems 4.1 and 4.2 in Section 4.1 are applicable to client assignment in the tree topology. According to Theorem 4.2, in an optimal assignment, if any client is assigned to a server through a path that involves a node u , then no client would be assigned to other servers through paths that involve u . As a result, the clients outside subtree T_u can be assigned to servers inside T_u only if no client inside T_u is assigned to any server outside T_u . In addition, if multiple clients outside T_u are assigned to servers inside T_u , they must be all assigned to the same server inside T_u . Similarly, the clients inside subtree T_u can be assigned to servers outside T_u only if no client outside T_u is assigned to any server inside T_u . Therefore, we classify the generalized problem into three types defined as follows. To facilitate presentation, we refer to an assignment of the clients inside subtree T_u as a T_u -client assignment. We shall also use $|T_u|$ to refer to the number of nodes in the subtree T_u .

Definition 4.1 *Assume that a client is located at each node in a tree network T and there are a set of servers S in T . Let u be a node in T .*

- (i) *Suppose that no client outside T_u is assigned to any server inside T_u . Given a number x ($1 \leq x \leq |T_u|$), among all T_u -client assignments that assign x clients inside T_u to servers outside T_u , the problem of finding a T_u -client assignment that minimizes $\mathcal{C}(E_u)$ is referred to as the (u, x) -problem (Figure 4.18(a)).*
- (ii) *Suppose that no client inside T_u is assigned to any server outside T_u . Given a server s inside T_u and a number x ($1 \leq x \leq |T| - |T_u|$), assuming that x clients outside T_u are assigned to server s , the problem of finding a T_u -client assignment that minimizes $\mathcal{C}(E_u)$ is referred to as the (u, x, s) -problem (Figure 4.18(b)).*

- (iii) Suppose that no client outside T_u is assigned to any server inside T_u , and no client inside T_u is assigned to any server outside T_u . The problem of finding a T_u -client assignment that minimizes $\mathcal{C}(E_u)$ is referred to as the $(u, 0)$ -problem (Figure 4.18(c)).

Figure 4.18: Generalized problems.

Suppose r is the root of the entire tree T . Then, no client is outside T_r , and E_r is the set of all links in T . So, the $(r, 0)$ -problem is the original client assignment problem in tree T .

4.4.2 Recurrences for Dynamic Programming

 Figure 4.19: T_u and its subtrees.

We first show that for each internal node u in tree T , an optimal solution to the (u, x) -, (u, x, s) - or $(u, 0)$ -problem can be obtained by a combination of the optimal solutions to

some subproblems. Suppose that node u has k children $c_1, c_2, \dots, c_k$ as shown in Figure 4.19. Since $E_u = \{(u, p(u))\} \cup \bigcup_{i=1}^k E_{c_i}$, it follows that

$$\mathcal{C}(E_u) = \mathcal{C}(\{(u, p(u))\}) + \sum_{i=1}^k \mathcal{C}(E_{c_i}). \quad (\text{Eq. 4.19})$$

Note that $\mathcal{C}(\{(u, p(u))\})$ is essentially $d(u, p(u))$ times the number of link $(u, p(u))$'s occurrences in the interaction paths between all pairs of clients. Given a (u, x) -, (u, x, s) - or $(u, 0)$ -problem, we show that there are a fixed number of $(u, p(u))$'s occurrences, which is independent of the T_u -client assignment.

In the (u, x) -problem, all clients in the entire tree T can be classified into three groups: $|T| - |T_u|$ clients are outside T_u and are assigned to servers outside T_u (abbreviated as Out-to-Out clients); x clients are inside T_u and are assigned to servers outside T_u (abbreviated as In-to-Out clients); and $|T_u| - x$ clients are inside T_u and are assigned to servers inside T_u (abbreviated as In-to-In clients). The interaction paths between Out-to-Out clients do not contain the link $(u, p(u))$. Similarly, the interaction paths between In-to-In clients do not contain the link $(u, p(u))$. Each interaction path between two In-to-Out clients or between an In-to-Out client and an In-to-In client contains the link $(u, p(u))$ twice. In addition, each interaction path between an Out-to-Out client and an In-to-In or In-to-Out client contains the link $(u, p(u))$ once. As a result, the total number of $(u, p(u))$'s occurrences is

$$\begin{aligned} & 2\left(\frac{x(x+1)}{2} + x \cdot (|T_u| - x)\right) + |T_u| \cdot (|T| - |T_u|) \\ &= |T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T_u| - x^2 + x. \end{aligned} \quad (\text{Eq. 4.20})$$

In the (u, x, s) -problem, all clients in the entire tree T can be classified into three groups: $|T| - |T_u| - x$ clients are outside T_u and are assigned to servers outside T_u (abbreviated as Out-to-Out clients); x clients are outside T_u and are assigned to servers inside T_u (abbreviated as Out-to-In clients); and $|T_u|$ clients are inside T_u and are assigned to servers inside T_u (abbreviated as In-to-In clients). Each interaction path between two Out-to-In clients and

each interaction path between an Out-to-In client and an Out-to-Out client contain the link $(u, p(u))$ twice. In addition, each interaction path between an In-to-In client and an Out-to-In or Out-to-Out client contains the link $(u, p(u))$ once. As a result, the total number of $(u, p(u))$'s occurrences in the (u, x, s) -problem is

$$\begin{aligned} & 2\left(\frac{x(x+1)}{2} + x \cdot (|T| - |T_u| - x)\right) + |T_u| \cdot (|T| - |T_u|) \\ &= |T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T| - 2x \cdot |T_u| - x^2 + x. \end{aligned} \quad (\text{Eq. 4.21})$$

In the $(u, 0)$ -problem, each interaction path between a client inside T_u and a client outside T_u contains the link $(u, p(u))$ once. The other interaction paths do not contain the link $(u, p(u))$. Therefore, the total number of $(u, p(u))$'s occurrences in the $(u, 0)$ -problem is

$$|T_u| \cdot (|T| - |T_u|) = |T_u| \cdot |T| - |T_u|^2. \quad (\text{Eq. 4.22})$$

Thus, the numbers of $(u, p(u))$'s occurrences in (Eq. 4.20), (Eq. 4.21) and (Eq. 4.22) are all fixed numbers for the respective problems. Therefore, in the (u, x) -, (u, x, s) - and $(u, 0)$ -problems, the objective of minimizing $\mathcal{C}(E_u)$ is equivalent to minimizing $\sum_{i=1}^k \mathcal{C}(E_{c_i})$.

Let $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ be the optimal T_u -client assignments to the (u, x) -, (u, x, s) - and $(u, 0)$ -problems respectively. It follows from the above analysis that all T_{c_i} -client assignments used in $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ are also optimal assignments to their corresponding subproblems for respective subtrees T_{c_i} . Otherwise, if there exist better T_{c_i} -client assignments, then replacing the T_{c_i} -client assignments in $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ with these better T_{c_i} -client assignments reduces $\mathcal{C}(E_{c_i})$ and thus $\mathcal{C}(E_u)$, which contradicts to the optimality of $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$.

Now we show how to compute the optimal assignments. We start by solving the client assignment problem for the case where servers are located at the leaf nodes only. We extend the solution to the general case where servers can be located at any nodes in the tree in Section 4.4.3.

To facilitate presentation, for each node u in T , we represent the optimal assignments $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ as the sets of pairs of clients and their assigned servers (i.e., $\{(c, s) \mid c \in T_u, s \in S, c \text{ is assigned to } s\}$). We also let $l(u, x)$, $l(u, x, s)$ and $l(u, 0)$ represent the values of $\mathcal{C}(E_u)$ for the optimal assignments $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$ respectively.

The optimization problems are trivial if u is a leaf node in T . If $u \notin S$ (i.e., there is no server at u), the client at u must be assigned to a server outside T_u . Thus, among all optimization problems of T_u , only the $(u, 1)$ -problem is a valid problem. $A(u, 1)$ assigns u to a server outside T_u , and $l(u, 1) = (|T| + 1) \cdot d(u, p(u))$ is a fixed value regardless of u 's assigned server. If $u \in S$ (i.e., there is a server at u), then only the (u, x, u) - and $(u, 0)$ -problems are valid. It is easy to obtain $A(u, x, u) = \{(u, u)\}$, $l(u, x, u) = (x(x+1) + 2x(|T| - x - 1) + (|T| - 1)) \cdot d(u, p(u))$; and $A(u, 0) = \{(u, u)\}$, $l(u, 0) = (|T| - 1) \cdot d(u, p(u))$.

For each non-leaf node u in T , we must examine the solutions to all possible subproblems in order to determine which solutions to use in the optimal assignments $A(u, x)$, $A(u, x, s)$ and $A(u, 0)$.

4.4.2.1 Recursive Solution to (u, x) -Problem

Figure 4.20: A recursive solution to the (u, x) -problem.

In the (u, x) -problem, x clients inside T_u are assigned to servers outside T_u . Since the paths from these clients to their assigned servers all go through node u , according to Theorem 4.2 in Section 4.1, these x clients must be assigned to the same server outside T_u , and according to Corollary 4.1 in Section 4.1, the client at node u must be one of these x clients. In addition, no client can be assigned to any other server through paths involving u . Thus, in the optimal assignment $A(u, x)$, no client outside each subtree T_{c_i} is assigned to servers inside T_{c_i} . As shown in Figure 4.20, for each subtree T_{c_i} ($1 \leq i \leq k$), if the T_{c_i} -client assignment used in $A(u, x)$ does not assign any client inside T_{c_i} to servers outside T_{c_i} , then it must be an optimal solution to the $(c_i, 0)$ -problem. Otherwise, if the T_{c_i} -client assignment used in $A(u, x)$ assigns x_i ($x_i \geq 1$) clients inside T_{c_i} to servers outside T_{c_i} , it must be an optimal solution to the (c_i, x_i) -problem. Moreover, based on Theorem 4.2, these x_i clients must be part of the x clients inside T_u that are assigned to a server outside T_u . Therefore, to solve the (u, x) -problem, we need to consider the $(c_i, 0)$ -problem and the (c_i, x_i) -problems ($1 \leq x_i \leq |T_{c_i}|$) for each subtree T_{c_i} . All $A(c_i, x_i)$ s selected to be used in $A(u, x)$ must satisfy $\sum_{i=1}^k x_i = x - 1$, where “1” represents the client at u . Therefore, following equation (Eq. 4.19) and expression (Eq. 4.20), the recurrence for the (u, x) -problem is given by

$$l(u, x) = (|T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T_u| - x^2 + x) \cdot d(u, p(u)) + \min_{\substack{0 \leq x_i \leq |T_{c_i}| \\ \sum_{i=1}^k x_i = x-1}} \left\{ \sum_{i=1}^k l(c_i, x_i) \right\}. \quad (\text{Eq. 4.23})$$

It is easy to see that for any node u and number x , the value of $l(u, x)$ is independent of the server outside T_u to which the x clients are assigned. Thus, we represent the assigned server of the x clients by a special symbol $\mathbb{O}$ in $A(u, x)$. Let x_i^* denote the value of x_i that produces the minimum $\sum_{i=1}^k l(c_i, x_i)$ in the above recurrence. Then, the optimal assignment $A(u, x)$ is given by

$$A(u, x) = \{(u, \mathbb{O})\} \cup \bigcup_{i=1}^k A(c_i, x_i^*).$$

The symbol $\mathbb{O}$ would be replaced by a specific server later in the recurrences for the (u, x, s) - and $(u, 0)$ -problems in Sections 4.4.2.2 and 4.4.2.3.

If all possible values of x_i s are examined exhaustively in the computation of recurrence (Eq. 4.23), the computation still has an exponential asymptotic complexity. The term $\min_{0 \leq x_i \leq |T_{c_i}|, \sum_{i=1}^k x_i = x-1} \{\sum_{i=1}^k l(c_i, x_i)\}$ in recurrence (Eq. 4.23) can be computed efficiently as follows. Let $f_1(j, x')$ be the minimum value of $\sum_{i=1}^j l(c_i, x_i)$ subject to the constraint $\sum_{i=1}^j x_i = x'$, i.e.,

$$f_1(j, x') = \min_{\substack{0 \leq x_i \leq |T_{c_i}| \\ \sum_{i=1}^j x_i = x'}} \left\{ \sum_{i=1}^j l(c_i, x_i) \right\},$$

where $1 \leq j \leq k$, and $0 \leq x' \leq x-1$. It is obvious that

$$\begin{cases} f_1(1, x') = l(c_1, x'), \\ f_1(j, x') = \min_{\substack{0 \leq x_j \leq \\ \min(x', |T_{c_j}|)}} \{f_1(j-1, x' - x_j) + l(c_j, x_j)\}. \end{cases}$$

Thus, starting from $j = 1$, we can compute all $f_1(j, x')$ s in an increasing order of j . Then, $l(u, x)$ can be computed with $f_1(k, x-1)$.

4.4.2.2 Recursive Solution to (u, x, s) -Problem

Figure 4.21: A recursive solution to the (u, x, s) -problem.

In the (u, x, s) -problem, as shown in Figure 4.21, x clients outside T_u are assigned to the server s inside T_u . The paths from these clients to server s must go through node u .

Suppose that server s is located inside a subtree $T_{c_{t(s)}}$ ($1 \leq t(s) \leq k$). Then, according to Theorem 4.2 in Section 4.1, no client inside $T_{c_{t(s)}}$ should be assigned to servers outside $T_{c_{t(s)}}$, and according to Corollary 4.1 in Section 4.1, the client at node u must also be assigned to server s . In addition, in the optimal assignment $A(u, x, s)$, if any client in the other subtrees T_{c_i} ($i \neq t(s)$) is assigned to servers inside $T_{c_{t(s)}}$, it must be assigned to server s as well. Assuming there are a total of w such clients in $A(u, x, s)$, then the $T_{c_{t(s)}}$ -client assignment used in $A(u, s, x)$ must be an optimal solution to the $(c_{t(s)}, x + 1 + w, s)$ -problem, where “1” represents the client at u . For each of the other subtrees T_{c_i} ($i \neq t(s)$), based on Theorem 4.2, there cannot be any client outside T_{c_i} assigned to servers inside T_{c_i} in $A(u, x, s)$. If the T_{c_i} -client assignment used in $A(u, x, s)$ does not assign any client inside T_{c_i} to servers outside T_{c_i} , then it must be an optimal solution to the $(c_i, 0)$ -problem. Otherwise, if the T_{c_i} -client assignment used in $A(u, x, s)$ assigns x_i ($x_i \geq 1$) clients inside T_{c_i} to servers outside T_{c_i} , it must be an optimal solution to the (c_i, x_i) -problem. In addition, these x_i clients must be assigned to server s inside $T_{c_{t(s)}}$. Therefore, for each subtree T_{c_i} that does not contain s , we need to consider the $(c_i, 0)$ -problem and the (c_i, x_i) -problems ($1 \leq x_i \leq |T_{c_i}|$). All $A(c_i, x_i)$ s selected to be used in $A(u, x, s)$ must satisfy $\sum_{i \neq t(s)} x_i = w$. Therefore, following equation (Eq. 4.19) and expression (Eq. 4.21), the recurrence for the (u, x, s) -problem is given by

$$\begin{aligned}
 l(u, x, s) = & (|T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T| - 2x \cdot |T_u| - x^2 + x) \cdot d(u, p(u)) \\
 & + \min_{\substack{\forall i \neq t(s), \\ 0 \leq x_i \leq |T_{c_i}|}} \left\{ l(c_{t(s)}, x + 1 + \sum_{i \neq t(s)} x_i, s) + \sum_{i \neq t(s)} l(c_i, x_i) \right\}. \quad (\text{Eq. 4.24})
 \end{aligned}$$

Recall that in the optimal assignment $A(c_i, x_i)$ to the (c_i, x_i) -problem, x_i clients inside T_{c_i} are assigned to a server outside T_{c_i} that is indicated by the symbol $\mathbb{O}$ and yet to be determined. Now, if $A(c_i, x_i)$ is selected to be used in $A(u, x, s)$, these x_i clients should be assigned to server s . We shall denote by $A(c_i, x_i)(s/\mathbb{O})$ the T_{c_i} -client assignment produced by replacing $\mathbb{O}$ with s in $A(c_i, x_i)$. Let x_i^* denote the value of x_i that produces the minimum

$l(c_{t(s)}, x+1 + \sum_{i \neq t(s)} x_i, s) + \sum_{i \neq t(s)} l(c_i, x_i)$ in the above recurrence. Then, the optimal assignment $A(u, x, s)$ is given by

$$A(u, x, s) = \{(u, s)\} \cup A(c_{t(s)}, x+1 + \sum_{i \neq t(s)} x_i^*, s) \cup \bigcup_{i \neq t(s)} A(c_i, x_i^*)(s/\mathbb{O}).$$

The term $\sum_{i \neq t(s)} l(c_i, x_i)$ in recurrence (Eq. 4.24) can be computed efficiently as follows. We first fix $\sum_{i \neq t(s)} x_i$ at a value z where $0 \leq z \leq \sum_{i \neq t(s)} |T_{c_i}|$. Then, the value of $l(c_{t(s)}, x+1 + \sum_{i \neq t(s)} x_i, s)$ is fixed accordingly. Let $f_2(j, x')$ be the minimum value of $\sum_{1 \leq i \leq j, i \neq t(s)} l(c_i, x_i)$ subject to the constraint $\sum_{1 \leq i \leq j, i \neq t(s)} x_i = x'$, i.e.,

$$f_2(j, x') = \min_{\substack{0 \leq x_i \leq |T_{c_i}| \\ \sum_{1 \leq i \leq j, i \neq t(s)} x_i = x'}} \left\{ \sum_{\substack{1 \leq i \leq j, \\ i \neq t(s)}} l(c_i, x_i) \right\}, \quad (\text{Eq. 4.25})$$

where $1 \leq j \leq k$ and $0 \leq x' \leq z$. Then,

$$\begin{cases} f_2(1, x') = l(c_1, x'), \\ f_2(j, x') = \begin{cases} \min_{\substack{0 \leq x_j \leq \\ \min(x', |T_{c_j}|)}} \{f_2(j-1, x' - x_j) + l(c_j, x_j)\}, & \text{if } j \neq t(s), \\ f_2(j-1, x'), & \text{if } j = t(s). \end{cases} \end{cases}$$

Starting from $j = 1$, the values of all $f_2(j, x')$ s can be computed in an increasing order of j . Then, the value of $l(u, x, s)$ subject to $\sum_{i \neq t(s)} x_i = z$ can be computed with $l(c_{t(s)}, x+1+z, s)$ and $f_2(k, z)$.

Considering all possible z values, we have

$$\begin{aligned} l(u, x, s) &= (|T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T| - 2x \cdot |T_u| - x^2 + x) \cdot d(u, p(u)) \\ &\quad + \min_{0 \leq z \leq \sum_{i \neq t(s)} |T_{c_i}|} \{l(c_{t(s)}, x+1+z, s) + f_2(k, z)\}. \end{aligned} \quad (\text{Eq. 4.26})$$

4.4.2.3 Recursive Solution to $(u, 0)$ -Problem

The $(u, 0)$ -optimization problem is similar to the (u, x, s) -optimization problem in that the client at node u must be assigned to a server s inside T_u , and for each subtree T_{c_i} , the clients inside T_{c_i} that are assigned to servers outside T_{c_i} should also be assigned to the same server

 Figure 4.22: A recursive solution to the $(u, 0)$ -problem.

s as the client at u does. However, different from the (u, x, s) -optimization problem where s is given, in the $(u, 0)$ -optimization problem s should run through all servers inside T_u . For each server s inside T_u , we consider the $(c_{t(s)}, x_{t(s)}, s)$ -problem for the subtree $T_{c_{t(s)}}$ that contains server s , and consider the $(c_i, 0)$ -problem and the (c_i, x_i) -problems ($1 \leq x_i \leq |T_{c_i}|$) for each subtree T_{c_i} ($i \neq t(s)$) that does not contain server s . All $A(c_i, x_i)$ s selected to be used in $A(u, 0)$ must satisfy $1 + \sum_{i \neq t(s)} x_i = x_{t(s)}$, where “1” represents the client at u . Therefore, following equation (Eq. 4.19) and expression (Eq. 4.22), the recurrence for the $(u, 0)$ -problem is given by

$$\begin{aligned}
 l(u, 0) &= (|T_u| \cdot |T| - |T_u|^2) \cdot d(u, p(u)) \\
 &+ \min_{\substack{s \in S \cap T_u, \forall i \neq t(s), \\ 0 \leq x_i \leq |T_{c_i}|}} \left\{ l(c_{t(s)}, 1 + \sum_{i \neq t(s)} x_i, s) + \sum_{i \neq t(s)} l(c_i, x_i) \right\}. \quad (\text{Eq. 4.27})
 \end{aligned}$$

Let x_i^* and s^* denote the values of x_i and s that produce the minimum $l(c_{t(s)}, 1 + \sum_{i \neq t(s)} x_i, s) + \sum_{i \neq t(s)} l(c_i, x_i)$ in the above recurrence. Then, $A(u, 0)$ is given by

$$A(u, 0) = \{(u, s^*)\} \cup A(c_{t(s^*)}, 1 + \sum_{i \neq t(s^*)} x_i^*, s^*) \cup \bigcup_{i \neq t(s^*)} A(c_i, x_i^*)(s^*/\odot).$$

We use a similar approach to previous recurrence computation for computing the term $\sum_{i \neq t(s)} l(c_i, x_i)$ in recurrence (Eq. 4.27). We first fix the server s . By employing the function $f_2(j, x')$ defined in (Eq. 4.25), the value of $l(u, 0)$ for a given s is given by

$$f_3(s) = (|T_u| \cdot |T| - |T_u|^2) \cdot d(u, p(u)) + \min_{0 \leq z \leq \sum_{i \neq t(s)} |T_{c_i}|} \{l(c_{t(s)}, 1 + z, s) + f_2(k, z)\}.$$

Considering all possible servers s , we have

$$l(u, 0) = \min_{s \in S \cap T_u} f_3(s).$$

4.4.2.4 Time Complexity Analysis

Starting from the subproblems where u is a leaf node, we can solve all (u, x) -, (u, x, s) - and $(u, 0)$ -optimization problems by a postorder traversal of T . The time complexity of our dynamic programming solution is analyzed as follows. Let K be the maximum degree of all nodes in the entire tree T , and $|S|$ be the number of servers. For each node u , in the (u, x) -problem, the time complexity to compute each $f_1(j, x')$ is $O(x')$. Since $1 \leq j \leq K$, $0 \leq x' \leq x - 1$ and $1 \leq x \leq |T_u|$, there are a total of $K \cdot |T_u|$ entries in the form of $f_1(j, x')$ in all (u, x) -problems. Thus, the time complexity to compute all $f_1(j, x')$ s ($1 \leq j \leq K$, $0 \leq x' < |T_u|$) is

$$O(K \cdot |T_u|^2) = O(K \cdot |T|^2).$$

This is the time complexity to solve all (u, x) -problems for a given node u .

For each node u , in the (u, x, s) -problem, the time complexity to compute each $f_2(j, x')$ is $O(x')$. Since $1 \leq j \leq K$, $0 \leq x' \leq z$ and $0 \leq z \leq \sum_{i \neq t(s)} |T_{c_i}|$, there are a total of $K \cdot (1 + \sum_{i \neq t(s)} |T_{c_i}|)$ entries in the form of $f_2(j, x')$ in all (u, x, s) -problems with the same s . Thus, for a given server s , the time complexity to compute all $f_2(j, x')$ s ($1 \leq j \leq K$, $0 \leq x' \leq \sum_{i \neq t(s)} |T_{c_i}|$) is

$$O(K \cdot (\sum_{i \neq t(s)} |T_{c_i}|)^2) = O(K \cdot |T_u|^2).$$

On the other hand, since $1 \leq x \leq |T| - |T_u|$, the time complexity to compute all $l(u, x, s)$ s with the same s using equation (Eq. 4.26) is $O(|T_u| \cdot (|T| - |T_u|))$. Therefore, the overall time complexity to solve all (u, x, s) -problems for a given node u is given by

$$O(|S| \cdot (K \cdot |T_u|^2 + |T_u| \cdot (|T| - |T_u|))) = O(K \cdot |S| \cdot |T|^2).$$

Since the $(u, 0)$ -problem is solved by a similar approach to the (u, x, s) -problem, the time complexity to solve all $(u, 0)$ -problems for a given node u is also $O(K \cdot |S| \cdot |T|^2)$.

Note that there are a total of $|T|$ nodes in the entire tree T . Therefore, the total time complexity of our dynamic programming solution is given by

$$\begin{aligned} &O(|T| \cdot (K \cdot |T|^2 + K \cdot |S| \cdot |T|^2 + K \cdot |S| \cdot |T|^2)) \\ &= O(K \cdot |S| \cdot |T|^3). \end{aligned}$$

In our algorithm, the optimal client assignments to all problems of each subtree T_u are computed from the optimal assignments in subtrees rooted at u 's children. Therefore, the proposed dynamic programming algorithm can be implemented in a distributed manner by having each node compute the optimal client assignments in its local subtree and send them to its parent on a level-by-level basis.

4.4.3 Extensions

With some simple modifications based on Theorem 4.1 in Section 4.1, the dynamic programming solution for the case in which servers are located at the leaf nodes only can be directly extended to the general case where servers can be located at any nodes in the tree. First, for each node $u \notin S$ (i.e., there is no server at u), we do not need to consider the (u, x, s) -problems where the path from u to s goes through another server. Theorem 4.1 indicates that such server s cannot be assigned any client outside subtree T_u . Similarly, in the recurrence for the $(u, 0)$ -problem, there is no need for server s to run through the servers where the paths from u to these servers go through other servers. Second, for each

node $u \in S$ (i.e., there is a server at u), we do not need to consider any (u, x) -problem since Theorem 4.1 implies that no client inside T_u should be assigned to any server outside T_u . Moreover, Theorem 4.1 also implies that if any client outside T_u is assigned to a server inside T_u , it can be assigned only to the server at u . Therefore, for the (u, x, s) -problems, we need to consider only those with $s = u$. In the (u, x, u) - and $(u, 0)$ -problems, since the clients inside each subtree T_{c_i} that are assigned to servers outside T_{c_i} should all be assigned to the server at u , the recurrences (Eq. 4.24) and (Eq. 4.27) can be simplified as

$$l(u, x, s) = (|T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T| - 2x \cdot |T_u| - x^2 + x) \cdot d(u, p(u)) + \min_{0 \leq x_i \leq |T_{c_i}|} \left\{ \sum_{i=1}^k l(c_i, x_i) \right\},$$

and

$$l(u, 0) = (|T_u| \cdot |T| - |T_u|^2) \cdot d(u, p(u)) + \min_{0 \leq x_i \leq |T_{c_i}|} \left\{ \sum_{i=1}^k l(c_i, x_i) \right\}.$$

The time complexity for computing the optimal client assignment in a general tree remains $O(K \cdot |S| \cdot |T|^3)$.

Next, we consider the case where clients are located at only a subset of nodes in the tree. By defining $|T|$ and $|T_u|$ to be the numbers of clients in the tree T and the subtree T_u respectively, the recurrences for all the three generalized problems remain the same if u is a client or a server. If u is neither a client nor a server, extending these recurrences to the (u, x) -, (u, x, s) - and $(u, 0)$ -problems is also straightforward: we only need modify the part of a recurrence where u was counted as a client. For example, in the (u, x) -problem, if u is neither a client nor a server, all $A(c_i, x_i)$ s selected to be used in $A(u, x)$ should satisfy $\sum_{i=1}^k x_i = x$ instead of $\sum_{i=1}^k x_i = x - 1$. Thus, the recurrence becomes

$$l(u, x) = (|T_u| \cdot |T| - |T_u|^2 + 2x \cdot |T_u| - x^2 + x) \cdot d(u, p(u)) + \min_{\substack{0 \leq x_i \leq |T_{c_i}| \\ \sum_{i=1}^k x_i = x}} \left\{ \sum_{i=1}^k l(c_i, x_i) \right\},$$

and the optimal assignment $A(u, x)$ is given by

$$A(u, x) = \bigcup_{i=1}^k A(c_i, x_i^*).$$

We can modify the recurrences to the (u, x, s) - and $(u, 0)$ -problems in similar ways. The time complexity for computing the optimal client assignment in the entire tree T is $O(K \cdot |S| \cdot |T|^2 \cdot |V|)$, where $|V|$ is the total number of nodes in T , and $|T|$ is the number of clients in T .

4.5 Summary

In this chapter, we have investigated the client assignment problem for discrete DIAs. We have proved that this problem is NP-complete. Then, three heuristic algorithms have been proposed with their approximation ratios theoretically analyzed and their performance experimentally evaluated using real Internet latency data. The results show that: (1) the proposed *Modify-Assignment* and *Distributed-Modify Assignment* algorithms significantly outperform the intuitive *Nearest-Server Assignment* algorithm for different server placements; (2) these modification-based algorithms generally require each client to perform just few assignment modifications; and (3) *Distributed-Modify Assignment* adapts well to the dynamics of both client participation and network latency. We have also developed a polynomial time algorithm for computing the optimal client assignment in tree topologies.

Chapter 5

Client Assignment for Continuous Distributed Interactive Applications

This chapter investigates the client assignment problem for continuous DIAs (Definition 3.2). Section 5.1 derives the hardness results. Section 5.2 proposes three heuristic assignment algorithms and analyzes their approximation ratios. The experimental setup and results are discussed in Section 5.3. Finally, Section 5.4 summarizes this chapter.

5.1 Hardness Results

In this section, we analyze the hardness of the client assignment problem for continuous DIAs (Definition 3.2). We first prove that the problem is NP-complete.

Theorem 5.1 *The client assignment problem for continuous DIAs is NP-complete.*

Proof Consider a candidate solution for an instance of the client assignment problem in its decision version with an integer bound L . The length of the interaction path between each pair of clients can be computed in polynomial time. Thus, computing the maximum length of interaction paths between all client pairs and comparing it with the bound L can be done in polynomial time. Therefore, the client assignment problem is in NP.

We show that the client assignment problem is NP-complete by a polynomial time reduction from the minimum set cover problem which is known to be NP-complete [34]. The

decision version of the minimum set cover problem is defined as follows: given a finite set P and a collection $\mathbb{Q}$ of its subsets, and a positive integer $K \leq |\mathbb{Q}|$, find out whether $\mathbb{Q}$ contains a set cover for P of size at most K , i.e., whether there exists a subset $\mathbb{Q}' \subseteq \mathbb{Q}$ with $|\mathbb{Q}'| \leq K$ such that $\bigcup_{Q \in \mathbb{Q}'} Q = P$. We focus on the non-trivial instances of the minimum set cover problem in which $K > 1$ and the collection $\mathbb{Q}$ does not contain the universe set P .

distance matrix:

		s_1^1	s_2^1	s_3^1	s_1^2	s_2^2	s_3^2
$P = \{p_1, p_2, p_3\}$							
$\mathbb{Q} = \{Q_1, Q_2, Q_3\}$							
$Q_1 = \{p_1\}$	c_1	1	$\frac{3}{2} + \frac{x}{2}$	$\frac{3}{2} + \frac{x}{2}$	1	$\frac{3}{2} + \frac{x}{2}$	$\frac{3}{2} + \frac{x}{2}$
$Q_2 = \{p_2, p_3\}$	c_2	$\frac{3}{2} + \frac{x}{2}$	1	$\frac{3}{2} + \frac{x}{2}$	$\frac{3}{2} + \frac{x}{2}$	1	$\frac{3}{2} + \frac{x}{2}$
$Q_3 = \{p_3\}$	c_3	$\frac{3}{2} + \frac{x}{2}$	1	1	$\frac{3}{2} + \frac{x}{2}$	1	1
set cover:	s_1^1	0	$1 + x$	$1 + x$	1	1	1
$\mathbb{Q}' = \{Q_1, Q_2\}$	s_2^1	$1 + x$	0	$1 + x$	1	1	1
client assignment	s_3^1	$1 + x$	$1 + x$	0	1	1	1
corresponding to $\mathbb{Q}'$:	s_1^2	1	1	1	0	$1 + x$	$1 + x$
$c_1 \rightarrow s_1^1$	s_2^2	1	1	1	$1 + x$	0	$1 + x$
$c_2 \rightarrow s_2^2$	s_3^2	1	1	1	$1 + x$	$1 + x$	0
$c_3 \rightarrow s_2^2$							

Figure 5.1: Example of an instance of the minimum set cover problem with $K = 2$ and the corresponding distance matrix of the client assignment problem.

Let R be an instance of the minimum set cover problem. Suppose that set P contains n elements $p_1, p_2, \dots, p_n$ and collection $\mathbb{Q}$ contains m subsets $Q_1, Q_2, \dots, Q_m$. We first construct a network consisting of n clients and $m \cdot K$ servers (see Figure 5.1). The n clients are $c_1, c_2, \dots, c_n$, and each client c_i corresponds to one element p_i in set P . The servers are divided into K groups with each group containing m servers:

$$s_1^1, s_2^1, \dots, s_m^1; s_1^2, s_2^2, \dots, s_m^2; \dots; s_1^K, s_2^K, \dots, s_m^K,$$

where the super-script represents the group number, and the sub-script represents the server number in the group. Each server corresponds to one subset in $\mathbb{Q}$, and the j th server s_j^l of each group l corresponds to subset Q_j . The distance between a client c_i and a server s_j^l is 1 if element p_i belongs to subset Q_j , and is $3/2 + x/2$ ($x > 0$) otherwise. Besides, the distance

between two servers is 1 if they are in different groups, and is $1 + x$ if they are in the same group. An instance T of the client assignment problem is then constructed on the network by setting the bound L to 3. Thus, instance T is constructed in time polynomial to the size of instance R . In the following, we show that $\mathbb{Q}$ contains a set cover $\mathbb{Q}'$ of size at most K for instance R if and only if there exists a client assignment with the maximum interaction path length at most 3 for instance T .

Suppose there exists a set cover $\mathbb{Q}'$ with $|\mathbb{Q}'| \leq K$, then a client assignment is constructed in $|\mathbb{Q}'|$ steps as follows. At each step, we consider one subset $Q_j \in \mathbb{Q}'$ and try to assign the clients that correspond to Q_j 's elements to the same server. We first choose an unused server group l such that no client has been assigned to any server in the group. Then, for each element $p_i \in Q_j$, client c_i is assigned to server s_j^l in group l if c_i has not been assigned to any server. Since we only assign clients to one server at each step, the total number of servers that are assigned clients is $|\mathbb{Q}'| \leq K$. Thus, it is guaranteed that an unused server group l can always be found at each step. In addition, since each element p_i belongs to at least one subset Q_j in $\mathbb{Q}'$, all clients must be assigned to servers after $|\mathbb{Q}'|$ steps. In the example of Figure 5.1, the set cover $\mathbb{Q}' = \{Q_1, Q_2\}$ contains two subsets of elements. Thus, the clients are assigned to two different servers s_1^1 and s_2^2 . In the assignment that we construct, each client c_i is assigned to one of the j th servers s_j^l of all groups only if p_i belongs to Q_j , which indicates that the distance between any client and its assigned server is 1. Since in any group, at most one server is assigned clients, the distance between any two different servers that are assigned clients is 1. Therefore, the maximum interaction path length in the constructed assignment is at most 3.

On the other hand, suppose there exists a client assignment with the maximum interaction path length not exceeding 3. Since the distance between any client and server is at least 1, the distance between any two servers that are assigned clients in this assignment cannot be longer than 1. This implies that in each group, there is at most one server that is

assigned clients, as the distance between any two servers in the same group is $1 + x$. Thus, the total number of servers that are assigned clients is at most the number of server groups, i.e., K . We construct a set cover $\mathbb{Q}'$ based on this client assignment by selecting a subset Q_j if and only if there exists at least one j th server s_j^l that is assigned clients. Then, $|\mathbb{Q}'| \leq K$. In addition, note that the distance between a client c_i and a server s_j^l is 1 if subset Q_j covers element p_i , and is $3/2 + x/2$ otherwise. If c_i is assigned to s_j^l where Q_j does not cover c_i , the length of the interaction path from c_i to itself is at least $3 + x$, which is not allowed in the aforementioned assignment. It follows that $\mathbb{Q}'$ must cover all elements in set P . Therefore, $\mathbb{Q}'$ is a set cover of size at most K .

Hence, the theorem is proven. $\square$

The above NP-completeness analysis also gives us the following impossibility result.

Theorem 5.2 *No polynomial-time algorithm for the client assignment problem can achieve an approximation ratio less than $4/3$ if the network latency satisfies the triangle inequality and any bounded approximation ratio otherwise, if $P \neq NP$.*

Proof Consider the networks that are constructed in the proof of Theorem 5.1. Since the subset collection $\mathbb{Q}$ does not contain the universe set P , any set cover must have a size of at least 2. Therefore, for each instance of the set cover problem in which there exists a set cover of size at most K , the maximum interaction path length of an optimal client assignment for the corresponding network is 3. On the other hand, any non-optimal assignment produces a maximum interaction path length at least $3 + x$. This is because in a non-optimal assignment, either a client is assigned to a server that has a distance $3/2 + x/2$ to it, or two servers in the same group (having a distance $1 + x$ from each other) are both assigned clients. In either case, the maximum interaction path length cannot be less than $3 + x$. Thus, there does not exist any assignment that produces a maximum interaction path length between 3 and $3 + x$. Therefore, if there exists a polynomial-time client assignment

algorithm with an approximation ratio less than $y = (3 + x)/3$, the algorithm is guaranteed to find the optimal assignment for the networks constructed from all instances of the set cover problem in which there exist set covers of size at most K . This implies that there exists a polynomial-time algorithm for the set cover problem, which contradicts to $P \neq NP$. If the triangle inequality is satisfied, x should not be larger than 1 (due to a triangle with sides 1, 1, and $1 + x$ formed by three servers) and therefore $y \leq 4/3$. Otherwise, y can be arbitrarily large as x goes towards infinity.

Hence, the theorem is proven. □

5.2 Heuristic Algorithms

A brute-force client assignment algorithm is computationally expensive even with small numbers of clients and servers. In this section, we present three heuristic client assignment algorithms. Similar to Section 4.2, the computation of these algorithms is based on the network latencies between clients and servers, which can be obtained with existing tools like ping and King [38]. Thus, these algorithms are generic and not tied to any particular routing strategy or the triangle inequality.

5.2.1 Nearest-Server Assignment

The first algorithm is the intuitive *Nearest-Server Assignment*. This algorithm can be implemented by having each client measure the network latencies between itself and all servers, and select the server with the lowest latency as its assigned server. The computational complexity for each client is hence $O(|S|)$. While this *Nearest-Server Assignment* reduces the client-to-server latencies, it could significantly increase the latencies between the assigned servers of different clients, and thus make the interactivity far worse than optimum.

When assuming that the network latency satisfies the triangle inequality, we can show that *Nearest-Server Assignment* has a tight approximation ratio of 3.

Theorem 5.3 *Nearest-Server Assignment has an approximation ratio of 3 for networks with the triangle inequality.*

Proof This proof is in spirit similar to a proof presented in [42]. Consider two clients u and v , whose nearest servers are n_u and n_v respectively. Suppose in an optimal assignment, client u is assigned to server o_u and client v is assigned to server o_v . Naturally, we have $d(u, n_u) \leq d(u, o_u)$ and $d(v, n_v) \leq d(v, o_v)$. By the triangle inequality, it follows that:

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

and

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

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

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

This result shows that the interaction path length for each client pair in *Nearest-Server Assignment* is within 3 times of that for the client pair in an optimal assignment. Therefore, the maximum interaction path length in *Nearest-Server Assignment* must be within 3 times of that in an optimal assignment.

Hence, the theorem is proven. □

Figure 5.2: The approximation ratio 3 of *Nearest-Server Assignment* is tight.

The approximation ratio of 3 is tight. Figure 5.2 gives an example in which there are two clients c_1, c_2 and three servers s, s_1, s_2 . The distances between both clients and server s are a , and the distance from client c_1 to server s_1 and the distance from client c_2 to server s_2 are $a - \varepsilon$, where $\varepsilon > 0$. In *Nearest-Server Assignment*, c_1 is assigned to s_1 , and c_2 is assigned to s_2 . So, the maximum interaction path length is $6a - 4\varepsilon$. It is obvious that the optimal assignment should assign both clients to server s , and its maximum interaction path length is $2a$. The ratio between the two maximum interaction path lengths $6a - 4\varepsilon$ and $2a$ can be made arbitrarily close to 3 when ε approaches 0.

Figure 5.3: Unbounded approximation ratio of *Nearest-Server Assignment* when the network latency does not satisfy the triangle inequality.

In the absence of the triangle inequality [53], *Nearest-Server Assignment* cannot achieve any bounded approximation ratio. Similar to Figure 4.5 in Section 4.2.1, Figure 5.3 shows an example in which the performance of *Nearest-Server Assignment* can be arbitrarily worse than that of the optimal assignment. In this example, c_1, c_2 are clients, and s_1, s_2, s_3, s_4 are servers. Suppose that $r > 4$. The maximum interaction path length in *Nearest-Server Assignment* is $r + 2$, while that in the optimal assignment is 6. Thus, the ratio between the two results can be made arbitrarily large as r goes towards infinity.

5.2.2 Greedy Assignment

The second algorithm *Greedy Assignment* adopts a greedy approach to assign clients iteratively, starting with an empty assignment. In each step, the algorithm considers all the possibilities of assigning an unassigned client to a server. If a client c is selected to be assigned to a server s , then all unassigned clients that are not farther from s than c are also assigned to s as this would not increase the maximum interaction path length. Let Δn be the number of new clients assigned to s in this way, and Δl be the increase in the maximum interaction path length due to these new assignments. To minimize the amortized increase in the maximum interaction path length, we use $\Delta l / \Delta n$ as the cost metric for selecting which client to be assigned to which server. In each step, among all possible pairs of unassigned client and server, the pair (c, s) resulting in the minimum cost $\Delta l / \Delta n$ is selected and the corresponding clients are then assigned to s . The algorithm terminates when all clients have been assigned to servers.

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

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

where $2d(c, s)$ is the interaction path length from c to itself and C' is the set of clients already assigned. For each server s , the term $\max_{b \in C'} \{ d(s, s_A(b)) + d(s_A(b), b) \}$ is independent of client c , so its calculation can be shared among all unassigned clients. The pseudo code of

Greedy Assignment is presented in Algorithm 2. *Greedy Assignment* is suited for centralized implementation due to its need for global knowledge about the distances between clients and servers.

Algorithm 2: The *Greedy Assignment* algorithm

```

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

```

In the preprocessing stage, sorting lists L_s for all servers and calculating the indexes of all clients in the lists can be done in $O(|S||C| \log |C|)$ time. Suppose all clients are assigned to servers in m steps. To calculate the time complexity of each step, we divide it into three

stages. Stage 1 (lines 9 to 21) is to find the pair of client and server with the minimum $\Delta l/\Delta n$. For each server, the time complexity of line 11 is $O(|C|)$. Then, the calculation of $\Delta l/\Delta n$ can be done in $O(1)$ time for each unassigned client and hence in $O(|C|)$ time for all unassigned clients. Thus, the total time complexity of stage 1 in each step is $O(|S||C|)$. Stage 2 (lines 22 to 25) is to add the new assignments of clients. The time complexity of this stage in each step is $O(|C|)$. Stage 3 (lines 26 to 31) is to update the indexes of unassigned clients in lists L_s by removing newly assigned clients. The time complexity of this stage in each step is $O(|S||C|)$. So, the total time complexity for one step is $O(|S||C|)$. Therefore, the overall time complexity of *Greedy Assignment* is $O(|S||C| \log |C| + m|S||C|)$, or $O(|S||C|^2)$ in the worst case since $m \leq |C|$.

Theorem 5.4 presents the approximation ratio of *Greedy Assignment* when assuming that the network latency satisfies the triangle inequality.

Theorem 5.4 *Greedy Assignment has an approximation ratio of $O(\log(n))$ for networks with the triangle inequality, where n is the number of clients.*

Proof We number all the clients according to the order in which they are assigned by *Greedy Assignment* as $c_1, c_2, \dots, c_n$, breaking ties arbitrarily. For each client c_i , let g_i be the assigned server of c_i in *Greedy Assignment*, and o_i be the assigned server of c_i in an optimal assignment. Denote the cost of assigning c_i in *Greedy Assignment* by $\text{cost}(c_i)$.

Let us examine the step of *Greedy Assignment* in which a client c_e is assigned. Consider the unassigned client c_r that is farthest from server o_e at the beginning of this step. If we assign c_r to o_e in this step, all unassigned clients including c_e would also be assigned to o_e . Let n' be the total number of unassigned clients at the beginning of this step, and l' be the increase in the maximum interaction path length caused by assigning c_r to o_e . Then, the cost of assigning c_r to o_e is l'/n' , which should not be less than the actual cost of assigning

c_e to g_e in *Greedy Assignment* since *Greedy Assignment* always chooses the assignment with the minimum cost. Therefore, we have

$$\text{cost}(c_e) \leq \frac{l'}{n'}.$$

We first analyze the value of l' . After assigning c_r to o_e , if c_r is not involved in the longest interaction path, then the maximum interaction path length should not increase, i.e., $l' = 0$. If c_r is involved in the longest interaction path, two cases need to be considered. First, if the new longest interaction path is the interaction path from c_r to itself, we have $l' \leq 2 \cdot d(c_r, o_e)$, since the increase l' cannot be larger than the length of the new longest interaction path itself. It follows from the triangle inequality that

$$\begin{aligned} l' &\leq 2 \cdot (d(c_r, o_r) + d(o_r, o_e)) \\ &\leq 2 \cdot (d(c_r, o_r) + d(o_r, o_e) + d(o_e, c_e)). \end{aligned}$$

Note that $d(c_r, o_r) + d(o_r, o_e) + d(o_e, c_e)$ is the length of an interaction path from c_r to c_e in the optimal assignment, which should not be larger than the maximum interaction path length in the optimal assignment (denoted by OPT). Therefore, we have $l' \leq 2 \cdot OPT$. Second, if the new longest interaction path is not the interaction path from c_r to c_r , it must be the interaction path from c_r to a client c_x that is already assigned to a server g_x in some previous step of *Greedy Assignment*. This is because all the clients assigned together with c_r to o_e in the current step are not farther from o_e than c_r . Let L be the maximum interaction path length before c_r is assigned to o_e . It is obvious that $L \geq 2 \cdot d(g_x, c_x)$. Thus, we have

$$\begin{aligned} l' &= d(c_r, o_e) + d(o_e, g_x) + d(g_x, c_x) - L \\ &\leq d(c_r, o_e) + d(o_e, g_x) - d(g_x, c_x). \end{aligned}$$

Again, by the triangle inequality, it follows that

$$\begin{aligned} l' &\leq d(c_r, o_r) + d(o_r, o_e) + d(o_e, o_x) + d(o_x, c_x) + d(c_x, g_x) - d(g_x, c_x) \\ &= d(c_r, o_r) + d(o_r, o_e) + d(o_e, o_x) + d(o_x, c_x) \\ &\leq d(c_r, o_r) + d(o_r, o_e) + d(o_e, c_e) + d(c_e, o_e) + d(o_e, o_x) + d(o_x, c_x). \end{aligned}$$

Note that $d(c_r, o_r) + d(o_r, o_e) + d(o_e, c_e) \leq OPT$ and $d(c_e, o_e) + d(o_e, o_x) + d(o_x, c_x) \leq OPT$ since they are the lengths of two interaction paths in the optimal assignment. As a result, we have $l' \leq 2 \cdot OPT$. In summary, in either case, l' is bounded by $2 \cdot OPT$.

For the value of n' , since c_e is the e th client assigned by *Greedy Assignment*, there are at least $n - e + 1$ unassigned clients in the network before c_e is assigned. Thus, we have $n' \geq n - e + 1$. Therefore,

$$\text{cost}(c_e) \leq \frac{l'}{n'} \leq \frac{2 \cdot OPT}{n - e + 1}.$$

Since the above inequality holds for all clients, we obtain

$$\begin{aligned} \sum_{i=1}^n \text{cost}(c_i) &\leq \sum_{i=1}^n \frac{2 \cdot OPT}{n - i + 1} \\ &= OPT \cdot 2 \sum_{i=1}^n \frac{1}{n - i + 1} \\ &= OPT \cdot O(\log(n)). \end{aligned}$$

The left side of the above inequality is exactly the maximum interaction path length produced by *Greedy Assignment*.

Hence, the theorem is proven. □

Figure 5.4: The approximation ratio of *Greedy Assignment* is asymptotically tight.

The above approximation ratio is asymptotically tight. Figure 5.4 presents an example network with n clients $c_1, c_2, \dots, c_n$ and $2n$ servers $s_1, s_2, \dots, s_n, s'_1, s'_2, \dots, s'_n$. The first step of *Greedy Assignment* must assign client c_1 alone to server s_1 , which has a cost of 2.

Then, c_2 is assigned alone to s_2 with a cost of $2n/(n-1)$, since assigning c_2 to any other server would result in larger cost. By the same logic, each c_i is assigned to s_i in *Greedy Assignment*. Thus, the maximum interaction path length is $2n \sum_{i=1}^n 1/i$. The optimal assignment should assign each c_i to s'_i , and has a maximum interaction path length of $(2 + \varepsilon)n$. Therefore, the ratio between the two results approaches $\sum_{i=1}^n 1/i = O(\log(n))$ as ε approaches 0.

In the absence of the triangle inequality, the example in Figure 5.3 also shows that *Greedy Assignment* has unbounded approximation ratio.

5.2.3 Distributed-Modify Assignment

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

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

To perform the assignment modification, the coordinator selects a server s that is involved in a longest interaction path and sends a request to it that includes the current D value. Then, server s attempts to reduce D by modifying the assignment of its client c that introduces the longest interaction path. First, server s broadcasts to all the other servers the identifier of c and the longest distance $l(s)$ to its assigned clients excluding c . On receiving

the information, each of the other servers s' measures its distance to c , and computes the maximum length of interaction paths involving c assuming c is assigned to it, which is given by $L(s') = \max_{s''} \{d(c, s') + d(s', s'') + l(s'')\}$. Then, each server s' sends the result $L(s')$ back to server s . If $\min_{s'} L(s') < D$, s reassigns c to the server s^* with the minimum $L(s^*)$. In this case, servers s and s^* update their longest distances to their clients and broadcast these distances to all the other servers if they are changed. Finally, the coordinator recalculates D . If D cannot be reduced after the coordinator has tried all the servers involved in the longest interaction path(s), the algorithm terminates.

Figure 5.5: An example in which changing the assigned servers of two clients simultaneously increases the maximum interaction path length.

If the coordinator selects two or more servers to perform assignment modifications concurrently, the maximum interaction path length is not guaranteed to reduce because the calculation of each assignment modification is based on the assumption that the assigned servers of other clients remain unchanged. Figure 5.5 gives an example. Suppose that clients c_1 and c_2 are initially assigned to servers s_1 and s_2 respectively, so that the maximum interaction path length is 30. If c_1 (or c_2) changes its assigned server to s_3 (or s_4), the maximum interaction path length would be reduced to 25. However, if both clients change their assigned servers, the interaction path length between c_1 and c_2 would become 40, which is even longer than the maximum interaction path length of the initial assignment. Therefore, the coordinator selects only one server at a time for assignment modification.

In each assignment modification, the computation of $L(s')$ by each server s' has a time complexity of $O(|S|)$. The time complexity for server s to find s^* with the minimum $L(s^*)$

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

Figure 5.6: Unbounded approximation ratio of *Distributed-Modify Assignment*.

Distributed-Modify Assignment has unbounded approximation ratio if it starts with an arbitrary initial assignment, even for networks with the triangle inequality. As an example, Figure 5.6 shows a network containing two clients and two servers. Suppose that $r > 1$. Initially, both clients are assigned to server s_1 and the maximum interaction path length is $2r$. It would increase if any client changes its server to s_2 . Thus, the algorithm would terminate with no assignment modification performed. However, it is obvious that both clients should be assigned to server s_2 in the optimal assignment so that the maximum interaction path length is 2. Thus, the ratio between the two results can be made arbitrarily large as r approaches infinity. On the other hand, if *Distributed-Modify Assignment* takes *Nearest-Server Assignment* as the initial assignment, the resultant assignment cannot be worse than the latter since the assignment modification can only reduce the maximum interaction path length. Our experiments in Section 5.3 show that *Distributed-Modify Assignment* normally outperforms *Nearest-Server Assignment* significantly.

5.2.4 Dealing With Server Capacity Constraints

Similar to Section 4.2.4, we adapt each proposed assignment algorithm to deal with server capacity constraints in this section.

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

The approximation ratios previously analyzed for “uncapacitated” assignment algorithms are not applicable to “capacitated” assignment algorithms. *Distributed-Modify Assignment* has unbounded approximation ratio even without server capacity limitation. Thus, the same is also true when the server capacity is limited. In the following, we present the tight approximation ratios of the “capacitated” *Nearest-Server Assignment* and the “capacitated” *Greedy Assignment*, when assuming that the network latency satisfies the triangle inequality.

We first analyze the approximation ratio of the “capacitated” *Nearest-Server Assignment* algorithm. For each client c , let $s_N(c)$ denote its assigned server in the “capacitated” *Nearest-Server Assignment*, and $s_O(c)$ denote its assigned server in an optimal “capacitated” assignment.

Lemma 5.1 *For each client c , there must exist a client c' such that $d(c, s_N(c)) \leq d(c, s_O(c'))$. Here c' may be the same as c .*

Proof Assume on the contrary that $d(c, s_N(c)) > d(c, s_O(c'))$ for all clients c' (including $c' = c$). This implies that the assigned servers of all clients in the optimal “capacitated” assignment are all different from server $s_N(c)$. Note that the total capacity of the servers that are assigned clients in the optimal “capacitated” assignment is no less than the total number of clients. So, it is guaranteed that when assigning client c in the “capacitated” *Nearest-Server Assignment*, at least one of these servers is not saturated. Since client c is closer to all these servers than to $s_N(c)$, c cannot be assigned to $s_N(c)$ in the “capacitated” *Nearest-Server Assignment*, which contradicts to the definition of $s_N(c)$.

Hence, the lemma is proven. $\square$

Theorem 5.5 *The “capacitated” Nearest-Server Assignment has an approximation ratio of 5 for networks with the triangle inequality.*

Proof Consider any two clients u and v . According to Lemma 5.1, let u' be a client such that $d(u, s_N(u)) \leq d(u, s_O(u'))$, and v' be a client such that $d(v, s_N(v)) \leq d(v, s_O(v'))$. By the triangle inequality, the interaction path length between u and v in the “capacitated” *Nearest-Server Assignment* follows

$$\begin{aligned}
 & d(u, s_N(u)) + d(s_N(u), s_N(v)) + d(v, s_N(v)) \\
 & \leq d(u, s_N(u)) + d(u, s_N(u)) + d(u, v) + d(v, s_N(v)) + d(v, s_N(v)) \\
 & = 2d(u, s_N(u)) + d(u, v) + 2d(v, s_N(v)) \\
 & \leq 2d(u, s_O(u')) + d(u, v) + 2d(v, s_O(v')).
 \end{aligned}$$

Each of the terms $d(u, s_O(u')), d(u, v), d(v, s_O(v'))$ on the right side of the above inequality does not exceed the length of some interaction path in the optimal “capacitated” assignment because

$$d(u, s_O(u')) \leq d(u, s_O(u)) + d(s_O(u), s_O(u')) + d(s_O(u'), u');$$

$$d(u, v) \leq d(u, s_O(u)) + d(s_O(u), s_O(v)) + d(s_O(v), v);$$

$$d(v, s_O(v')) \leq d(v, s_O(v)) + d(s_O(v), s_O(v')) + d(s_O(v'), v').$$

Therefore, $d(u, s_O(u')), d(u, v)$ and $d(v, s_O(v'))$ are all not larger than the maximum interaction path length OPT in the optimal “capacitated” assignment. Thus,

$$d(u, s_N(u)) + d(s_N(u), s_N(v)) + d(v, s_N(v)) \leq 5 \cdot OPT. \quad (\text{Eq. 5.1})$$

Since inequality (Eq. 5.1) holds for any two clients u and v , it implies that the maximum interaction path length in the “capacitated” *Nearest-Server Assignment* must be within 5 times of that in an optimal “capacitated” assignment.

Hence, the theorem is proven. $\square$

Figure 5.7: The approximation ratio 5 of the “capacitated” *Nearest-Server Assignment* is tight.

Figure 5.7 shows an example illustrating that the approximation ratio 5 of the “capacitated” *Nearest-Server Assignment* is tight. In this network, the latencies from client c_{n+1} (c'_{n+1}) to servers s_{n+1} and s_{n+2} (s'_{n+1} and s'_{n+2}) are ϵ and 2 respectively. For each

$i = 1, 2, \dots, n$, the latencies from client c_i (c'_i) to servers s_i and s_{i+1} (s'_i and s'_{i+1}) are $1/n + \varepsilon$ and $1/n - \varepsilon$ respectively. For each $i, j = 1, 2, \dots, n + 1$, the latency between servers s_i and s'_j is 2. Assume that each server can accommodate one client only. Suppose that in *Nearest-Server Assignment*, the clients are assigned in the order of $c_1, c'_1, c_2, c'_2, \dots, c_{n+1}, c'_{n+1}$. Then, each pair of clients c_i and c'_i ($i = 1, 2, \dots, n$) are assigned to servers s_{i+1} and s'_{i+1} respectively. Eventually, clients c_{n+1} and c'_{n+1} can only be assigned to servers s_{n+2} and s'_{n+2} respectively. As a consequence, the maximum interaction path length in this assignment is $10 + 2\varepsilon$, which is the length of the interaction path between c_{n+1} and c'_{n+1} . The optimal assignment should assign c_i and c'_i to s_i and s'_i respectively ($i = 1, 2, \dots, n + 1$). This produces a maximum interaction path length of $2 + 2/n + 2\varepsilon$, which is between any c_i and c'_j ($i, j = 1, 2, \dots, n$). Thus, the ratio between the two results can be made arbitrarily close to 5 as n goes towards infinity and ε approaches 0.

Next, we analyze the approximation ratio of the “capacitated” *Greedy Assignment* algorithm.

Theorem 5.6 *The “capacitated” Greedy Assignment has an approximation ratio of $O(n)$ for networks with the triangle inequality, where n is the number of clients.*

Proof We number all the clients according to the order in which they are assigned by the “capacitated” *Greedy Assignment* as $c_1, c_2, \dots, c_n$, breaking ties arbitrarily. For each client c_i , let g_i and o_i be the assigned servers of c_i by the “capacitated” *Greedy Assignment* and an optimal “capacitated” assignment respectively, and let $\text{cost}(c_i)$ be the cost of assigning c_i to g_i in the “capacitated” *Greedy Assignment*.

Let us consider the step of the “capacitated” *Greedy Assignment* in which a client c_e is assigned. Let c_r be the unassigned client farthest to g_e at the beginning of this step. Denote by $l(c \rightarrow s)$ the increase of the maximum interaction path length introduced by assigning a client c to a server s in this step. We claim that there must exist a client c_x

such that $\text{cost}(c_e) \leq l(c_r \rightarrow o_x)$. Otherwise, suppose that $\text{cost}(c_e) > l(c_r \rightarrow o_i)$ for all clients c_i ($i = 1, 2, \dots, n$). Since $\text{cost}(c_e) \leq l(c_e \rightarrow g_e) \leq l(c_r \rightarrow g_e)$, we have $g_e \neq o_i$ for all clients c_i . Note that the total capacity of the servers that are assigned clients in the optimal assignment is no less than the total number of clients. Thus, at least one server o_y , which is different from g_e , is not saturated before assigning c_e . Since $\text{cost}(c_e) > l(c_r \rightarrow o_y)$, assigning c_r to o_y at this step would produce lower cost than assigning c_e to g_e , which contradicts to that *Greedy Assignment* always chooses the assignment with the minimum cost. Therefore, there exists a client c_x such that $\text{cost}(c_e) \leq l(c_r \rightarrow o_x)$. Similar to the analysis in the proof of Theorem 5.4, it can be shown that $l(c_r \rightarrow o_x) \leq 2 \cdot \text{OPT}$, where OPT is the maximum interaction path length in the optimal “capacitated” assignment. It follows that $\text{cost}(c_e) \leq 2 \cdot \text{OPT}$.

Since the above inequality holds for all clients, we have

$$\sum_{i=1}^n \text{cost}(c_i) \leq 2n \cdot \text{OPT} = \text{OPT} \cdot O(n).$$

Hence, the theorem is proven. $\square$

Figure 5.8: The approximation ratio of the “capacitated” *Greedy Assignment* is asymptotically tight.

The above approximation ratio is asymptotically tight. Figure 5.8 presents an example network with n clients $c_1, c_2, \dots, c_n$ and $2n$ servers $s_1, s_2, \dots, s_n, s'_1, s'_2, \dots, s'_n$. The capacity of each server is 1. The “capacitated” *Greedy Assignment* assigns each client c_i to server

s_i , producing a maximum interaction path length of $2 \sum_{i=1}^n i = n(n+1)$, while the optimal “capacitated” assignment is to assign each client c_i to server s'_i , resulting in a maximum interaction path length of $3n-1-2\varepsilon$. Therefore, the ratio between the two results approaches $n(n+1)/(3n-1-2\varepsilon) = O(n)$ as ε approaches 0.

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

5.3 Experimental Evaluation

5.3.1 Experimental Setup

We evaluate the performance of the proposed algorithms by simulation experiments making use of the same Meridian data set [1] as in Section 4.3. The network that we simulate is represented by a complete matrix of pair-wise latencies among 1,796 nodes. One client is assumed to be located at each node, and a certain number of servers are placed at selected nodes in the network. The proposed assignment algorithms are evaluated with various parameter settings of server number and capacity. In the default setting, there is no constraint on the server capacity. We use *Nearest-Server Assignment* as the initial assignment of *Distributed-Modify Assignment* in our experiments.

Similar to Section 4.3, we simulate three types of server placement: random, k -center, and k -median. In random server placement, for each parameter setting, we execute 1,000 simulation runs using 1,000 different sets of randomly placed servers. Unless stated otherwise, the average performance of these 1,000 simulation runs is plotted together with the 90th and 10th percentile results. For k -center and k -median server placements, we adopt a greedy k -center algorithm [26] and a greedy k -median algorithm [63] for placing servers in the experiments.

We develop a theoretical lower bound on the maximum interaction path length to quantify the relative performance of the assignment algorithms. The interaction path between

any two clients c and c' has a length of

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

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

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

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

5.3.2 Performance Comparison of Different Algorithms

First, we compare the performance of the client assignment algorithms without assuming any limitation on the server capacity. Figure 5.9 shows the normalized interactivity of each algorithm as a function of the number of servers in the network. As seen from Figure 5.9(a), the average performance of *Nearest-Server Assignment* under random server placement is significantly worse than that of the other two algorithms for all server numbers tested. Figure 5.10 shows the detailed cumulative distribution of the normalized interactivity for the 1,000 simulation runs of 80 randomly placed servers. The normalized interactivity produced by *Nearest-Server Assignment* exceeds 2 in over 100 simulation runs and exceeds 3 in over 50 runs. In contrast, the other two algorithms hardly result in any normalized interactivity above 2. Under k -center and k -median server placements, *Nearest-Server Assignment* also performs far worse than the other two heuristics as shown in Figures 5.9(b) and 5.9(c). The above results imply that intuitively assigning each client to its nearest server is not the most effective way of enhancing interactivity, even under carefully planned server placements.

(a) Random placement

 (b) k -center placement

 (c) k -median placement

Figure 5.9: Performance of client assignment algorithms for different numbers of servers.

Figure 5.10: Cumulative distribution of the normalized interactivity for networks with 80 randomly placed servers.

Comparing *Greedy Assignment* and *Distributed-Modify Assignment*, their performance is generally close to each other. The interactivity produced by both algorithms is within normally 40% of the lower bound, which means that it is within at most the same percentage of the optimum.

Figure 5.11: Cumulative distribution of the number of assignment modifications performed by *Distributed-Modify Assignment*.

Note that *Distributed-Modify Assignment* proceeds with assignment modifications until the maximum interaction path length cannot be further reduced. Figure 5.11 shows the distribution of the numbers of assignment modifications performed in the simulation. We

Table 5.1: Computation times for networks with 80 servers

Client assignment algorithm	CPU time (second)		
	Random	k -center	k -median
<i>Nearest-Server</i>	0.79	0.80	0.81
<i>Greedy</i>	3.26	2.97	2.22
<i>Distributed-Modify</i>	1.42	1.19	1.20

count each attempt made to change the assigned server of a client involved in a longest interaction path as one assignment modification, even if it is decided not to change the assigned server of the client after calculation. As seen from Figure 5.11, the algorithm performs fewer than 50 assignment modifications in 94% of the simulation runs. In fact, the average number of assignment modifications performed is less than 18. Under k -center and k -median placements, the numbers of assignment modifications are about 55 and 43 on average. These results imply that only a small proportion of clients need to modify their servers in the execution of *Distributed-Modify Assignment*.

Table 5.1 reports the computation times of the algorithms under 80 servers of various placements in our simulation performed on a PC with a Pentium Xeon 3.2 GHz CPU and 6GB RAM. It can be seen that *Distributed-Modify Assignment* has much shorter computation time than *Greedy Assignment*.

5.3.3 Impact of Server Capacity

Next, we evaluate the “capacitated” assignment algorithms under various server capacity constraints. We refer to the maximum number of clients that is allowed to be assigned to a server as the server capacity. All servers are assumed to have the same capacity. The “capacitated” *Nearest-Server Assignment* is used as the initial assignment of the “capacitated” *Distributed-Modify Assignment*. Figure 5.12 shows the normalized interactivity for different server capacities when 80 servers are placed in the network. Note that the theoretical lower bound does not change with the server capacity since its calculation assumes unlimited server

(a) Random placement

 (b) k -center placement

 (c) k -median placement

Figure 5.12: Performance of client assignment algorithms for different server capacities.

capacity. As expected, the interactivity performance produced by all algorithms deteriorates as the server capacity decreases. This is because when the server capacity is limited, the algorithms may not be able to assign clients to their most preferred servers. Comparing different algorithms, *Nearest-Server Assignment* and *Distributed-Modify Assignment* are generally less affected by server capacity constraints than *Greedy Assignment*. This implies that the client assignments of *Nearest-Server Assignment* and *Distributed-Modify Assignment* are more balanced among servers. The relative performance of the three assignment algorithms remains similar over different server capacities unless the server capacity is severely limited. When the server capacity is severely limited, all the algorithms produce similar interactivity because there is little space for optimizing client assignment to improve interactivity. Similar results are also observed for other server numbers.

5.3.4 Dynamic Scenarios

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

5.3.4.1 Adaptivity to Client Joining/Leaving

We first consider the scenario in which clients can join and leave the network freely. We use the same methodology as in Section 4.3.5.1 to model the joining and leaving patterns of clients. We randomly assign each client two weights w_j and w_l ($0 < w_j, w_l < 1$). w_l indicates the chance of the client leaving the DIA when it is currently participating, and w_j indicates the chance of the client joining the DIA when it is not participating at present. Small w_j and w_l mean that the client is inclined to keep its current state of participation unchanged, while large w_j and w_l imply that the client tends to change its participation state frequently. The client participation behavior is then modeled by a sequence of events, each of which could be either a join event or a leave event. Each event is randomly generated based on the weights and participation states of all clients. We start from a network in which no client is participating and simulate a sequence of 10,000 events. It has been observed from Figure 4.14 that about half of the clients participate in the stable state.

To adapt to dynamic client participation, *Nearest-Server Assignment* simply assigns each joining client to its nearest server. For *Distributed-Modify Assignment*, whenever a client joins the network, it is initially assigned to its nearest server. Then, for each join and leave event, based on the existing assignment, assignment modifications are performed until the maximum interaction path length cannot be further reduced. For *Greedy Assignment*, we record the clients assigned at each step and the cost of assigning them. When a participating client c leaves, the clients that are assigned before c in the previous execution are directly assigned to the same servers as in the previous execution. Then, the original *Greedy Assignment* is executed to assign the remaining clients. When a new client c joins, assume that in *Greedy Assignment*, clients are assigned in the same way as in the previous execution up to step $i - 1$. Let l_i be the maximum interaction path length and n_i be the number of unassigned clients before step i starts. Then, if client c is assigned at step i , the

increase in the maximum interaction path length cannot be less than $2 \cdot \min_{s \in S} d(c, s) - l_i$, and the number of clients assigned at this step cannot be larger than n_i . Thus, the cost of assigning c at step i cannot be less than $x_i = (2 \cdot \min_{s \in S} d(c, s) - l_i)/n_i$. If x_i is larger than the cost of step i of the previous execution, client c cannot be assigned at step i , which implies that the newly assigned clients at this step would be the same as in the previous execution. Therefore, instead of executing the original *Greedy Assignment* algorithm, we can assign clients in the same way as in the previous execution as long as x_i is larger than the cost of step i of the previous execution. After encountering the first x_i not exceeding the cost of step i , we start executing the original *Greedy Assignment* algorithm to assign the remaining clients.

Figure 5.13 shows the interactivity performance of each algorithm when 80 servers are placed in the network. The results presented for random server placement are those for one sample set of randomly placed servers. Other sets of servers tested show similar performance trends. As can be seen, the relative performance of all algorithms remains similar to the results in the static scenario. In general, *Distributed-Modify Assignment* consistently achieves the best interactivity among the three algorithms. Our statistics show that the average numbers of assignment modifications performed by *Distributed-Modify Assignment* at each join and leave event are just 1.95, 1.57 and 1.87 under the three server placements. This implies that performing just a few assignment modifications per event is sufficient to maintain decent interactivity performance, and thus, the computational and communication overheads introduced are very acceptable.

5.3.4.2 Adaptivity to Dynamics in Network Latency

The latency for any information to be transmitted across the network is affected by the network conditions and may vary with time. In this section, we evaluate the adaptivity of the algorithms to the variance of network latency by using the same PlanetLab-All-Pairs-Pings data set [2] as in Section 4.3.5.2. We simulate a network consisting of 188 nodes and

(a) One sample random placement

 (b) k -center placement

 (c) k -median placement

Figure 5.13: Performance of client assignment algorithms with dynamic client participation.

emulate the dynamics in network latency using the 95 sets of latency measurements on May 9, 2005. Again, a client is assumed to be located at each node.

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

Figure 5.14 presents the normalized interactivity of all algorithms over the day with 15 servers placed in the network. The k -center and k -median placements are calculated based on the first set of latency measurements. Similar to Section 4.3.5.2, this models the scenario in which the network latency information used for the deployment of servers is predicted by the first measurement. The theoretical lower bound on the maximum interaction path length is recomputed after each set of measurements for deriving the normalized interactivity. It can be seen from Figure 5.14 that *Distributed-Modify Assignment* produces better interactivity and performs much more stably than *Nearest-Server Assignment* under all server placements. The average numbers of assignment modifications performed after each latency measurement are just 5.73, 9.02 and 5.95 under the three server placements. Therefore, the computational and communication overheads are very acceptable. *Greedy Assignment* generally produces similar interactivity performance to *Distributed-Modify Assignment*. However, as stated earlier, *Greedy Assignment* is a centralized algorithm which would introduce much higher computation and communication overheads than *Distributed-Modify Assignment* in adapting the client assignment.

5.4 Summary

In this chapter, we have studied the client assignment problem for continuous DIAs. The problem is proven to be NP-complete. Three heuristic assignment algorithms are presented.

(a) One sample random placement

(b) k -center placement at initial latency measurement (c) k -median placement at initial latency measurement

Figure 5.14: Performance of client assignment algorithms with dynamic network latency.

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

Chapter 6

Conclusion

In recent years, advances in high-speed networking technologies and powerful computer processing capability have enabled rapid development of DIAs. In this thesis, we have investigated the problem of assigning clients to appropriate servers in an effective and efficient manner for enhancing the interactivity performance of distributed interactive computing. This chapter summarizes our research and contributions as well as discusses some future research directions.

In a typical configuration of DIAs, the application's state is calculated and maintained by servers, while clients are responsible for sending user-initiated operations and receiving the state updates. Due to the human-in-the-loop nature of DIAs, it is of crucial importance to improve the interactivity of DIAs for supporting graceful interactions among clients. Network latency is known as a major obstacle to enhancing interactivity in DIAs. Increasing geographical spreads of participants in large-scale DIAs are making distributed server deployment vital for combating the network latency. This latency-driven server distribution is desirable even when there are no limitations on the availability of server resources at one location.

In a distributed server architecture, the application's state is often replicated across a group of distributed servers, and each client is assigned to one server. When an operation is issued by a client, it is first sent to the server to which the client is assigned, and is then

forwarded to all the other servers. After the computation of the application's new state, state updates are sent by each server to its assigned clients. Thus, the interaction time between any pair of clients includes not only the network latencies from the clients to their assigned servers, but also the network latencies between the servers. These network latencies are directly influenced by the assignment of the clients to the servers. In addition, the interaction time is also affected by the consistency and fairness requirements of DIAs, both of which are important factors in providing decent interaction experience. To maintain consistency and provide fairness, artificial synchronization delays may be added into the interaction among clients, which in turn affect the interaction time in DIAs. These synchronization delays are also dependent on the assignment of the clients. Therefore, client assignment is an important issue to the interactivity of DIAs.

Based on whether the application's state changes due to the passing of time, DIAs can be classified into discrete DIAs and continuous DIAs. In the first part of the thesis, we have investigated the client assignment problem for discrete DIAs which change their states in response to user operations only. In the second part of the thesis, we have studied the client assignment problem for continuous DIAs which change the states not only in response to user operations but also because of the passing of time.

6.1 Contributions

The major contributions of the thesis are summarized as follows:

- We have mathematically modeled the interactivity performance of both discrete and continuous DIAs, taking the consistency and fairness requirements into consideration.

Based on the analysis, we have formulated the client assignment problems for the two types of DIAs as combinatorial optimization problems. Both problems are proved to be NP-complete via polynomial time reductions from the partition problem and the

set cover problem. We have further obtained an approximability result for the client assignment problem of continuous DIAs.

- For each client assignment problem, several heuristic assignment algorithms have been proposed, including both centralized and distributed algorithms. The approximation ratios of these algorithms have been theoretically analyzed and examples have been given to show the tightness of these ratios. We have also adapted each proposed assignment algorithm to deal with server capacity constraints when assuming that the servers have limited capacities. Some approximation ratios and tightness examples of the “capacitated” algorithms have been presented.
- The performance of the client assignment algorithms has been experimentally evaluated using real Internet latency data. The results show that: (1) our proposed *Greedy Assignment*, *Modify Assignment* and *Distributed-Modify Assignment* algorithms significantly outperform the intuitive *Nearest-Server Assignment* algorithm that assigns each client to its nearest server; (2) *Modify Assignment* and *Distributed-Modify Assignment* generally require just a small proportion of clients to perform assignment modifications for interactivity enhancement; (3) the *Distributed-Modify Assignment* algorithm shows good adaptivity to dynamics in both client participation and network latency.
- For the client assignment problem of discrete DIAs, we have shown that there exists a polynomial-time optimal solution for the special case of tree network topologies. An $O(K \cdot |S| \cdot |C|^2 \cdot |V|)$ algorithm is developed to compute the optimal client assignment, where K is the maximum degree of all nodes, $|S|$ is the number of servers, $|C|$ is the number of clients, and $|V|$ is the total number of nodes in the tree.

6.2 Future Research Directions

Many research issues can be further explored for interactivity enhancement in DIAs. It is also worth studying the extension of our algorithm design and analysis techniques to other application domains. In the following, we propose four future research directions.

6.2.1 Optimal Client Assignment for Continuous DIAs in Tree Topologies

We have formulated the client assignment problem for continuous DIAs and proved that it is NP-complete for general network topologies. However, there may exist polynomial-time optimal solutions for the special case of tree network topologies. In a tree topology, there is a unique path between each pair of nodes. If the longest interaction path of a client assignment is $\mathcal{L}$, the length of the path from any client to the mid-point of $\mathcal{L}$ should not exceed half the length of $\mathcal{L}$. In the future work, we plan to investigate a polynomial-time algorithm to compute the optimal client assignment for continuous DIAs in the tree topology by making use of this observation.

6.2.2 Server Placement

The server placement problem aims to find the locations of servers in order to optimize the interactivity of DIAs. As shown in Chapter 3, discrete and continuous DIAs have different formulations of the interaction time. Thus, the optimal placement of the servers is expected to be different for different types of DIAs. For example, suppose that in a full binary tree, the network latencies on all links are identical, and all clients are located at the leaf nodes. For continuous DIAs, since the interaction time can be formulated as the maximum length of interaction paths between all client pairs, we only need to place one server at the tree root in the optimal placement. Placing more servers at other nodes would not reduce the maximum interaction path length any further. However, for discrete DIAs, since the interaction time is formulated as the average length of interaction paths between all client pairs, this average

generally reduces if more servers are placed in the network. Assuming there are sufficient number of servers available, the optimal placement is to place one server at each internal node of the tree. Therefore, it is necessary to investigate the server placement problem for different types of DIAs separately.

When the objective is to minimize the average interaction path length, the server placement problem is related to the k -median problem. The difference is that, we must take into account the interserver latency as well in the server placement problem. Some existing research suggested that placing servers at the nodes with the highest node degrees may efficiently reduce the total client-to-server latency [69]. However, this approach is not guaranteed to obtain the optimal server placement. For example, in the network shown in Figure 6.1, $c_1, c_2, \dots, c_{12}$ are clients, and the network latency of each link is annotated above the link. Suppose that two servers are available and can be placed at any nodes in the network. Although the nodes with the highest node degrees are c_1 and c_5 , placing servers at c_1 and c_5 would introduce significant interaction times among clients c_{10} , c_{11} and c_{12} . Placing servers at c_9 and c_{10} could provide better overall interactivity. Thus, finding the optimal server placement for minimizing the average interaction path length is not a trivial task.

Figure 6.1: Example of the server placement problem.

When the objective is to minimize the maximum interaction path length, the server placement problem is related to the k -center problem. Similarly, the interserver latency needs to be considered, and it is not guaranteed to be optimal if we simply place servers at the nodes with the highest node degrees. One observation is that the lower bound of the maximum interaction path length is the length of the longest path between all client pairs. In the tree network topology, there is only one path between any two nodes. If we place a server at the midpoint along the longest path in the tree, the distance between any client to this server would not exceed half the length of the longest path. Then, by assigning all clients to this server, the maximum interaction path length is the length of the longest path. Therefore, the server placement problem in the tree network topology is trivial. However, it is difficult to find the longest path and there may exist multiple paths between two nodes in the general network topology, which implies that the server placement problem in general networks is still a challenging problem.

6.2.3 Inter-server Link Provisioning

In the distributed server architecture, servers can be interconnected with well-provisioned network links to enhance the interactivity. However, the number of well-provisioned links to set up may be limited due to deployment and maintenance costs. Therefore, we introduce an interserver link provisioning problem, which can be generally described as follows: given a fixed budget limiting the number of interserver links that can be provisioned, find the strategy to provision the interserver links to optimize the interactivity of DIAs. The solution to this problem should consider the locations of the clients and servers. For example, while it might be a good choice to provision the links that carry heavy communication traffic, the links with large network latency should also be provisioned. Again, the problem formulations could vary for different types of DIAs and different provisioning approaches are required to be developed.

6.2.4 Virtual Machine Placement in Distributed Clouds

Distributed clouds consist of a large number of data centers which are geographically distributed and interconnected over a wide-area network [3]. To serve user requests, virtual machines are required to be allocated in distributed clouds [7]. When a user makes a request to run an application in the cloud, the request specifies the number of virtual machines needed and the communication requirement among these virtual machines. In distributed clouds, user requests can be serviced from data centers close to them. This helps reduce network capacity needs and the latency of accesses. On the other hand, the latency in the communication between different data centers also has significant impact on the performance of users' applications. Therefore, similar to our client assignment problem, virtual machine placement should consider the distances between data centers as well as the distances from the data centers to the users. Since user requests may arrive and leave at any time, and the network latencies may vary as time progresses, dynamic scenarios also need to be handled in virtual machine placement. We expect that our proposed heuristics and analysis would be helpful to the study of virtual machine placement in distributed clouds.

Author's Publications

The following publications have been produced from this thesis research:

- **L. Zhang** and X. Tang. The client assignment problem for continuous distributed interactive applications: analysis, algorithms, and evaluation. Accepted to appear in *IEEE Transactions on Parallel and Distributed Systems*.
- **L. Zhang** and X. Tang. Optimizing client assignment for enhancing interactivity in distributed interactive applications. *IEEE/ACM Transactions on Networking*, vol.20, no.6, pp.1707–1720, Dec. 2012.
- **L. Zhang** and X. Tang. The client assignment problem for continuous distributed interactive applications. In *Proceedings of IEEE ICDCS 2011*, pp.203–214, June 2011.
- **L. Zhang** and X. Tang. Client assignment for improving interactivity in distributed interactive applications. In *Proceedings of IEEE INFOCOM 2011*, pp.3227–3235, Apr. 2011.

References

- [1] The meridian latency data set. Available at: <http://www.cs.cornell.edu/People/egs/meridian/>.
- [2] Planetlab all-pairs-pings. Available at: <http://pdos.lcs.mit.edu/~strib/>.
- [3] SCOPE Alliance. Telecom grade cloud computing. <http://scope-alliance.org/>.
- [4] A. Aggarwal, L. Anand, M. Bansal, N. Garg, N. Gupta, S. Gupta, and S. Jain. A 3-approximation for facility location with uniform capacities. In *Proceedings of IPCO 2010*, pages 149–162, 2010.
- [5] A. Agustina, F. Liu, S. Xia, H. Shen, and C. Sun. CoMaya: incorporating advanced collaboration capabilities into 3D digital media design tools. In *Proceedings of ACM CSCW 2008*, pages 5–8, 2008.
- [6] L. Ahmad, A. Boukerche, A. Al Hamidi, A. Shadid, and R. Pazzi. Web-based e-learning in 3D large scale distributed interactive simulations using HLA/RTI. In *Proceedings of IEEE IPDPS 2008*, pages 1–4, 2008.
- [7] M. Alicherry and T. Lakshman. Network aware resource allocation in distributed clouds. In *Proceedings of IEEE INFOCOM 2012*, pages 963–971, 2012.
- [8] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k -median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.
- [9] D. Bauer, S. Rooney, and P. Scotton. Network infrastructure for massively distributed games. In *Proceedings of ACM SIGCOMM NetGames 2002*, pages 36–43, 2002.
- [10] A. Bhambe, J. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *Proceedings of ACM SIGCOMM 2008*, pages 389–400, 2008.
- [11] C. Bouras, E. Giannaka, and T. Tsiatsos. Partitioning of Distributed Virtual Environments Based on Objects’ Attributes. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 72–75, 2007.
- [12] L. Briceño, H. Siegel, A. Maciejewski, Y. Hong, B. Lock, M. Teli, F. Wedyan, C. Panaccione, C. Klumph, K. Willman, and C. Zhang. Robust resource allocation in a massive multiplayer

- online gaming environment. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 232–239, 2009.
- [13] J. Brun, P. Boustead, and F. Safaei. Fairness and playability in online multiplayer games. In *Proceedings of the 2nd IEEE International Workshop on Networking Issues in Multimedia Entertainment*, 2006.
- [14] J. Brun, F. Safaei, and P. Boustead. Managing latency and fairness in networked games. *Communications of the ACM*, 49(11):46–51, 2006.
- [15] J. Brun, F. Safaei, and P. Boustead. Server topology considerations in online games. In *Proceedings of ACM SIGCOMM NetGames 2006*, 2006.
- [16] R. Burkard, M. Dell’Amico, and S. Martello. *Assignment problems*. Society for Industrial Mathematics, 2009.
- [17] W. Cai, S. Turner, B. Lee, and J. Zhou. An alternative time management mechanism for distributed simulations. *ACM Transactions on Modeling and Computer Simulation*, 15(2):109–137, 2005.
- [18] J. Campbell. Hub location and the p -hub median problem. *Operations Research*, 44(6):923–935, 1996.
- [19] C. Carlsson and O. Hagsand. DIVE-A multi-user virtual reality system. In *1993 IEEE Virtual Reality Annual International Symposium*, pages 394–400, 1993.
- [20] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza. Locality aware dynamic load management for massively multiplayer games. In *Proceedings of ACM PPOPP 2005*, pages 289–300, 2005.
- [21] K. Chen, P. Huang, and C. Lei. How sensitive are online gamers to network quality? *Communications of the ACM*, 49(11):34–38, 2006.
- [22] S. Choi and Y. Shavitt. Placing servers for session-oriented services. Technical Report WUCS-01-41, Washington University in St. Louis, 2001.
- [23] M. Chrobak, C. Kenyon, and N. Young. The reverse greedy algorithm for the metric k -median problem. *Information Processing Letters*, 97(2):68–72, 2006.
- [24] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.
- [25] E. Cronin, B. Filstrup, and A. Kurc. A distributed multiplayer game server system. Technical report, University of Michigan, 2001.
- [26] E. Cronin, S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. *IEEE Journal on Selected Areas in Communications*, 20(7):1369–1382, 2002.
- [27] E. Cronin, A. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, 23(1):7–30, 2004.

- [28] B. De Vleeschauwer, F. De Turck, B. Dhoedt, and P. Demeester. Server placement algorithms for the construction of a QoS enabled gaming infrastructure. In *Proceedings of ISCC 2005*, pages 185–190, 2005.
- [29] D. Delaney, T. Ward, and S. McLoone. On consistency and network latency in distributed interactive applications: A survey-Part I. *Presence: Teleoperators & Virtual Environments*, 15(2):218–234, 2006.
- [30] Y. Deng, R. Lau, et al. On delay adjustment for dynamic load balancing in distributed virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):529–537, 2012.
- [31] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the Internet. *IEEE Network Magazine*, 13(4):6–15, 1999.
- [32] J. Falby, M. Zyda, D. Pratt, and R. Mackey. NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation. *Computers & Graphics*, 17(1):65–69, 1993.
- [33] S. Ferretti, C. Palazzi, M. Rocchetti, M. Gerla, and G. Pau. Buscar el Levante por el Poniente: in search of fairness through interactivity in massively multiplayer online games. In *Proceedings of 2nd IEEE International Workshop on Networking Issues in Multimedia Entertainment*, 2006.
- [34] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman and Company, Calif, 1979.
- [35] L. Gautier, C. Diot, and J. Kurose. End-to-end transmission control mechanisms for multiparty interactive applications on the Internet. In *Proceedings of IEEE INFOCOM 1999*, pages 1470–1479, 1999.
- [36] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- [37] C. Greenhalgh, J. Purbrick, and D. Snowdon. Inside MASSIVE-3: flexible support for data consistency and world structuring. In *Proceedings of the 3rd International Conference on Collaborative Virtual Environments*, pages 119–127, 2000.
- [38] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 5–18, 2002.
- [39] D. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical programming*, 22(1):148–162, 1982.
- [40] M. Hori, T. Iseri, K. Fujikawa, S. Shimojo, and H. Miyahara. Scalability issues of dynamic space management for multiple-server networked virtual environments. In *Proceedings of the 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 200–203, 2001.
- [41] M. Hosseini, D. Ahmed, S. Shirmohammadi, and N. Georganas. A survey of application-layer multicast protocols. *IEEE Communications Surveys & Tutorials*, 9(3):58–74, 2007.

- [42] M. Iwasa, H. Saito, and T. Matsui. Approximation algorithms for the single allocation problem in hub-and-spoke networks and related metric labeling problems. *Discrete Applied Mathematics*, 157(9):2078–2088, 2009.
- [43] C. Jay, M. Glencross, and R. Hubbard. Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment. *ACM Transactions on Computer-Human Interaction*, 14(2), 2007.
- [44] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of IEEE INFOCOM 2004*, pages 96–107, 2004.
- [45] M. Korupolu, C. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000.
- [46] M. Kwok and J. Wong. Scalability analysis of the hierarchical architecture for distributed virtual environments. *IEEE Transactions on Parallel and Distributed Systems*, 19(3):408–417, 2008.
- [47] L. Lao, J. Cui, M. Gerla, and S. Chen. A scalable overlay multicast architecture for large-scale applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):449–459, 2007.
- [48] R. Lau. Hybrid load balancing for online games. In *Proceedings of ACM Multimedia 2010*, pages 1231–1234, 2010.
- [49] R. Lea, Y. Honda, K. Matsuda, and S. Matsuda. Community Place: architecture and performance. In *Proceedings of the 2nd Symposium on Virtual Reality Modeling Language*, pages 41–50, 1997.
- [50] K. Lee, B. Ko, and S. Calo. Adaptive server selection for large scale interactive online games. *Computer Networks*, 49(1):84–102, 2005.
- [51] Y. Lin, K. Guo, and S. Paul. Sync-MS: Synchronized messaging service for real-time multiplayer distributed games. In *Proceedings of IEEE ICNP 2002*, 2002.
- [52] J. Lui and M. Chan. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):193–211, 2002.
- [53] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee. Triangle inequality and routing policy violations in the Internet. In *Proceedings of PAM 2009*, pages 45–54, 2009.
- [54] M. Marzolla, S. Ferretti, and G. D’Angelo. Dynamic resource provisioning for cloud-based gaming infrastructures. *ACM Computers in Entertainment*. To appear.
- [55] M. Mauve, S. Fischer, and J. Widmer. A generic proxy system for networked computer games. In *Proceedings of ACM SIGCOMM NetGames 2002*, pages 25–28, 2002.
- [56] 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.
- [57] P. Morillo, J. Orduna, M. Fernandez, and J. Duato. Improving the performance of distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):637–649, 2005.

- [58] J. Müller, A. Gössling, and S. Gorlatch. On correctness of scalable multi-server state replication in online games. In *Proceedings of ACM SIGCOMM NetGames 2006*, 2006.
- [59] B. Ng, R. Lau, A. Si, and F. Li. Multiserver support for large-scale distributed virtual environments. *IEEE Transactions on Multimedia*, 7(6):1054–1065, 2005.
- [60] D. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, pages 111–120, 1995.
- [61] K. Pan, W. Cai, X. Tang, S. Zhou, and S. Turner. A hybrid interest management mechanism for peer-to-peer networked virtual environments. In *Proceedings of IEEE IPDPS 2010*, pages 1–12, 2010.
- [62] J. Pellegrino and C. Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *Proceedings of ACM SIGCOMM NetGames 2003*, pages 52–59, 2003.
- [63] L. Qiu, V. Padmanabhan, and G. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM 2001*, pages 1587–1596, 2001.
- [64] F. Safaei, P. Boustead, C. Nguyen, J. Brun, and M. Dowlatshahi. Latency-driven distribution: infrastructure needs of participatory entertainment applications. *IEEE Communications Magazine*, 43(5):106–112, 2005.
- [65] C. Savery, T. Graham, and C. Gutwin. The human factors of consistency maintenance in multiplayer computer games. In *Proceedings of ACM GROUP 2010*, pages 187–196, 2010.
- [66] S. Singhal and M. Zyda. *Networked virtual environments: design and implementation*. Addison-Wesley Reading, MA, 1999.
- [67] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, 1998.
- [68] D. Sun and C. Sun. Context-based operational transformation in distributed collaborative editing systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(10):1454–1470, 2009.
- [69] D. Ta and S. Zhou. A network-centric approach to enhancing the interactivity of large-scale distributed virtual environments. *Computer Communications*, 29(17):3553–3566, 2006.
- [70] D. Ta and S. Zhou. A two-phase approach to interactivity enhancement for large-scale distributed virtual environments. *Computer Networks*, 51(14):4131–4152, 2007.
- [71] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996.
- [72] D. Tang, L. Woo, and L. Bahl. Optimization of teleprocessing networks with concentrators and multiconnected terminals. *IEEE Transactions on Computers*, 100(27):594–604, 1978.
- [73] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez. Greening the Internet with nano data centers. In *Proceedings of ACM SIGCOMM CoNEXT 2009*, pages 37–48, 2009.

- [74] S. Webb and S. Soh. Adaptive client to mirrored-server assignment for massively multiplayer online games. In *Proceedings of MMCN 2008*, 2008.
- [75] S. Webb, S. Soh, and W. Lau. Enhanced mirrored servers for network games. In *Proceedings of ACM SIGCOMM NetGames 2007*, pages 117–122, 2007.
- [76] S. Zander, I. Leeder, and G. Armitage. Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 117–124, 2005.