Online Client Assignment in Dynamic Real-Time Distributed Interactive Applications

Seyhan Ucar, Huseyin Guler, Oznur Ozkasap
Department of Computer Engineering
Koc University, Istanbul, Turkey
[sucar, hguler, oozkasap]@ku.edu.tr

***Abstract*—Quality of user experience in Distributed Interactive Applications (DIAs) highly depends on the network latencies during the system execution. In DIAs, each user is assigned to a server and communication with any other client is performed through its assigned server. Hence, latency measured between two clients, called interaction time, consists of two components. One is the latency between the client and its assigned server, and the other is the inter-server latency, that is the latency between servers that the clients are assigned. In this paper, we investigate a real-time client to server assignment scheme in a DIA where the objective is to minimize the interaction time among clients. The client assignment problem is known to be NP-complete and heuristics play an important role in finding near optimal solutions. We propose two distributed heuristic algorithms to the online client assignment problem in a dynamic DIA system. We utilized real-time Internet latency data on the PlanetLab platform and performed extensive experiments using geographically distributed PlanetLab nodes where nodes can arbitrarily join/leave the system. The experimental results demonstrate that our proposed algorithms can reduce the maximum interaction time among clients up to 45% compared to an existing baseline technique.**

I. Introduction

Distributed Interactive Applications (DIAs) are network applications that enable interaction between clients geographically distributed around the world. Online games, military simulations and collaborative designs are some examples of DIAs [1]. In a DIA, minimizing the communication delay is a crucial objective that attracts more clients to join the system. The communication delay in DIA is defined as the time duration between when a client triggers an operation and when this operation is transferred to other clients [2]. Different architectures have been proposed to decrease the interaction time between clients [3] which can be classified into three groups, namely client-server, peer-to-peer and mirrored distributed server architectures.

In the client-server architecture, one server controls the application and each client connects to the system through that single entity. Consistency is one of the important advantages because each client is directly informed by the central server and each receives other clients' operations simultaneously. However, since clients can only connect to a central server, this server may become a bottleneck for the application. In peer-to-peer architecture, instead of using a central server, clients are connected to each other and share the workload among them. Workload sharing can be done in different ways such as partitioning the environment into regions, and assigning each of the regions to one client. However, the problem in the peer-to-peer system arises when the performance of a client is bad relative to the others. For example, in online games, clients may handle the processing of region assignments instead of receiving game updates which decreases the user satisfaction.

The last proposed architecture is the mirrored server architecture in which a predefined number of servers are connected to each other and each stores a replica of the application environment. Each client connects to one of the servers and all interaction among clients is done through clients' assigned servers [4]. When a client issues an operation, it first sends its action to its assigned server. Upon receiving the operation, each server computes the new state of the system and informs its connected clients.

In the mirrored server architecture, clients interact with other clients through their assigned servers. Interaction time is calculated as the sum of the latency between clients and their assigned servers, and the latency between assigned servers [5]. Client assignment plays a key role in determining the interaction time. A well-designed DIA must be able to assign an incoming client to a server in a way that the maximum interaction time between any client is minimized.

In [6], it is proved that the mirrored server architecture outperforms other architectures in a large scale distributed system. In this paper, the mirrored server architecture is used to minimize the interaction time among clients. Different from previous works, we utilize a real-time dynamic distributed interactive environment in which clients may join or leave the system at any time during the system execution. We propose two distributed algorithms for online version (i.e., the latency between nodes is not known beforehand) of the client assignment problem where the objective is to decide how to assign each client to servers so that the maximum interaction time clients is minimized. The original contribution of this paper is two-fold. First, to the best of our knowledge, our proposed *D-Nearest* and *D-GMIN* distributed algorithms are the first ones evaluated on real large-scale network test bed in an online adaptive manner where clients dynamically join or leave the system. Second, we utilize real-time Internet latency

data obtained on the PlanetLab[1] that constructs a continuous real-time distributed system.

The rest of the paper is organized as follows. We survey the related literature in Section II. Section III mathematically formulates the client assignment problem and describes the system model. Our online heuristic client assignment algorithms are proposed in Section IV. The experimental setup is presented in Section V. Section VI provides the main findings and comparative analysis of the proposed algorithms and the paper is concluded in Section VII.

II. Related Work

The literature presents few studies that directly address the client assignment problem in DIAs. In [7] and [8], authors propose mirrored server placement algorithm for content distribution networks (CDN). In these works, the objective is to serve clients in a fast manner by redirecting incoming clients into one of the mirrored servers. Given the set of servers, authors investigate the placement of these servers to maximize the performance. In contrast to CDN, in DIA rather than finding optimal geographical server placement the idea is to find optimal client to server assignments. Moreover, each client in DIA is connected to one server and clients interact with each other through their assigned servers.

In [5] and [9], authors prove that the client assignment problem is NP-complete and there is no polynomial time algorithm to find the optimal solution. For that reason, they propose four heuristic algorithms. In *Nearest-Server Assignment*, clients are greedily assigned to their nearest server. In *Longest-First-Batch Assignment*, the first client is assigned to its nearest server. All other clients which are not far away from this client are assigned to the same server since they will not increase the interaction delay. If that is not the case, then the client will be assigned to its nearest server and the interaction delay is updated accordingly. The *Greedy Assignment* works similar to *Longest-First-Batch Assignment* and the only difference is that they use a cost metric to decide which server to assign the client. In *Distributed-Greedy Assignment*, the process starts with the initial assignment and continues to modify client assignments until the point where maximum interaction path cannot be reduced further. They utilize the Meridian [10] internet latency data in the evaluation of their algorithms.

In [11] and [12], authors propose an approach to enhance the interactivity of DIAs by only considering the network latencies between client and server pairs. After the server placement, proposed algorithm uses the network latencies during the client assignment. However, as we show in Section VI, the inter-server latencies also play a critical role in improving the interactivity in DIAs.

In [13], the proposed solution is based on a virtual environment that is partitioned into several zones and each zone is controlled by a server. Clients in the same zone can interact with each other and clients can move to other zones as well. They propose two algorithms namely, *Initial Assignment* where zones are sorted based on the total weight of clients then assign the first zone to the first server, and *Refined Assignment* where they further reduce the Initial Assignment by reassigning the clients whose communication delay to their current server exceed a pre-defined threshold.

In [14], the assignment problem is mathematically modeled and an approximation algorithm is proposed. The study shows that finding the optimal client-server assignment with pre-defined requirements is NP-hard and relaxed convex optimization is proposed to find an approximate solution. The main idea behind the proposed optimization algorithm is to divide servers into two groups recursively until the point where no further split can be applied.

In [15], a partitioning algorithm is proposed to reduce the inconsistency in a multi-server distributed virtual environment. The main purpose is to efficiently distribute the network traffic generated by avatars among different servers in the system. By using the metric time-space inconsistency [16], the problem is formulated as a mixed integer programming problem. Alternating optimization is used to divide the problem into two sub-problems.

In [17], the authors investigate the update scheduling for distributed virtual environment (DVE). The key idea is to keep the DVE consistent where state updates are applied based on their potential impacts on the consistency. They propose three algorithms that utilize current network delays and estimate inconsistencies that may occur in future and show that the proposed algorithms significantly outperform the intuitive update algorithms. Different from our work, where we aim to find near optimal client to server assignments, they focus on how to schedule particular updates by using network capacity and delay.

In [18] and [19], the existing algorithms in [5] and [9] are modified to handle dynamic network conditions. Since the Meridian [10] set does not consider the latency variation over time, authors collect pairwise latency data from Planetlab-All-Pairs-Ping [20] over a one day period. By using collected Internet latency data, they experimentally evaluate the proposed algorithms with dynamic client join/leave. However, they still consider an offline version of the client assignment problem using latencies between clients and servers known beforehand hence not real-time. In contrast, we examine the online client assignment problem in this work.

III. Problem Formulation

In DIA networks, there exist several geographically distributed servers that are fully connected. Each client is assigned to a server and client communications or client operations in such systems are spread to all clients via their respective assigned server. For instance, consider an online multi-player game in which there are two clients named John (c_{John}) and Jane (c_{Jane}). An action performed by John is first sent to the assigned server of John that disseminates this information to other assigned servers in the system. Each assigned server, upon receiving an action, sends the

[1] http://www.planet-lab.org/

TABLE I
SYSTEM PARAMETERS

Parameters	Symbol
Number of clients	C
Number of servers	S
Client i	c_i
Server i	s_i
Capacity of server i	$capacity_{s_i}$
Decision variable whether c_i is assigned to s_j	$x_{i,j}$
Assigned server of client i	$s(c_i)$
Latency between entity i and entity j	$d(i,j)$

information to all of its clients. By this way, state change in the network is shared by all clients including John and Jane, and at the end, each client has the same state of the system, that is the system is in a consistent state. System parameters used in this section are listed in Table I.

$$x_{i,j} = \begin{cases} 1, & \text{if } c_i \text{ is assigned to } s_j, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Analyzing the above example introduces two notions of latencies. One of them exists between a client and its assigned server, and the other is the inter-server latency. The interaction path between client i and client j is given as d(c_i, s(c_i)) + d(s(c_i), s(c_j)) + d(s(c_j),c_j) where $s(c_i)$ is the assigned server of client i and d(i, j) is the latency between entity i and entity j. Our objective in such a DIA network is to minimize the maximum interaction path between each client pair as formulated below.

Our objective is to minimize

$$max(d(c_i, s(c_i)) + d(s(c_i), s(c_j)) + d(s(c_j), c_j)), \quad \forall c_i, c_j \in C \quad (2)$$

subject to

$$\sum_{i=1}^{C} x_{i,j} \leq capacity_{s_j}, \forall s_j \in S \quad (3)$$

IV. HEURISTIC ALGORITHMS

Since the client assignment problem is NP-complete, there is no polynomial time algorithm to find the optimal client-server assignment. Brute force approach, in which all possible assignment instances in the solution space are evaluated, lacks to find the optimal solution in reasonable time even for small number of clients and servers. For that reason, heuristic algorithms are preferred to find near optimal solutions to the client assignment problem. However, previous solutions fall into a major drawback, that is the dynamicity in real networks. All proposed solutions in the literature assume that clients and servers are always active in the system. On the contrary, in real networks and DIAs, it is highly likely for a client to leave the system for some time and then reconnect. Moreover, servers can fail and may not be able to respond the incoming client requests. Furthermore, performance of the existing solutions is evaluated using the delay measurements obtained by existing tools, such as Meridian [21] and MIT [22]. However, as shown in Fig. 1, latency between a random pair of nodes changes in time and there is no obvious pattern in the latency data to estimate future latencies. Thus, using such precalculated latency data is not realistic in real-time distributed interactive environments.

Our heuristic algorithms are designed to handle dynamic client join and leaves, and directly use real network latencies measured periodically between the utilized PlanetLab nodes. Before starting to assign clients to the servers, our algorithms first perform a latency measurement between all client and server pairs by simply exchanging hello packets. Latency measurements are saved and accessible by all servers, and all client assignments in our system are determined by the servers. In the rest of the section, we introduce our heuristic solutions to the client assignment problem. Notations used in these algorithms are given in Table II.

TABLE II
ALGORITHM NOTATIONS

Notations	Description
$INITIAL_{message}$	Initial Message of clients
REQ	Request Messages
REQ_{op}	Current Operation of Request Message
$SENDER_{req}$	Sender of Request Message
MAX_{path}	System Wide Maximum Interaction Path
S_{min}	Server with the Least Latency and Available Capacity
$S_{optimal}$	Server with the Least Latency
S_{c_i}	Assigned Server of Client i
$d(c_i, s_j)$	Latency between Client i and Server j
$Avail_{s_i}$	Available Capacity of Server i
$Wait_{s_i}$	Waiting List of Server i
c_{wait}	Waiting Client in the Waiting List
c_{req}	Client which generated the REQ
c_{list}	List of Connected Clients

A. Static Nearest-Server Algorithm (S-Nearest)

We use the *S-Nearest* algorithm as the baseline since it is one of the most widely used technique in distributed interactive environments. Based on the latency values in the servers, each newly joined client i is greedily assigned to its nearest server j unless that server is out of capacity (nearest does not necessarily mean the geographically nearest server but rather in terms of the network latency, $d(i,j)$). Otherwise, the client is assigned to the second nearest server and so on. When connecting to the system, each client selects a random server and sends a join request to that random server. Since all latency values are exchanged, join requests are forwarded to the nearest server of the newly joined client. Hereafter, that server is the assigned server of that client and all further communication of the client with the system will be performed through that server. Since for each client we have $|S|$ possible servers to select from, runtime complexity of *S-Nearest* algorithm is $O(|S|)$ for each client.

This algorithm has already been proposed in [9] and [5], and it is proved to have an approximation ratio of 3. However, that ratio only holds for a static system in which latency values

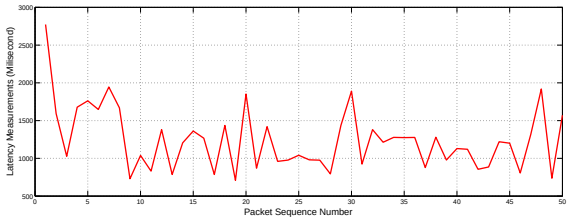

Fig. 1. Real-time latency values measured between two PlanetLab nodes (lsirextpc01.epfl.ch and planetlab01.tkn.tu-berlin.de) with 50 packet exchanges

between clients and servers stay unchanged. In our system, these latency values change in real-time as reported in Fig 1.

B. Dynamic Nearest-Server Algorithm (D-Nearest)

As an improvement to the *S-Nearest* algorithm, we propose a dynamic approach named *D-Nearest* as depicted in Algorithm-1 and Algorithm-2. Different from the static case, *D-Nearest* periodically calculates and exchanges latency values between the clients and their assigned servers. Updated latency values are also exchanged between servers so that each server has a global latency view of the system (Lines 1-15). Client assignments use the updated latencies by assigning each client to its nearest server as in the static case (Lines 19-35). In *D-Nearest*, we also introduce a novel approach by using waiting lists. If any client could not be assigned to its nearest server, our algorithm assigns the client to its second nearest server and additionally adds this client to the waiting list of its nearest server (Lines 30-31). Whenever a client leaves the system, its assigned server checks its own waiting list to see if there is any client that could not be assigned because its capacity was reached. If there is any, the server takes over the control of that client. Then, it sends a packet to the client indicating that its assigned server has changed and also sends another packet to the previous server to delete that client from its client list. Finally, the client is also removed from the waiting list since it is now connected to its nearest server (Algorithm 2). Our aim here is to keep the servers fresh in terms of client latency values to have a sustainable system with good performance.

C. Dynamic Distributed Greedy Minimum Delay (D-GMIN)

As *S-Nearest* and *D-Nearest* algorithms greedily utilize the minimum distanced servers to each client in the assignment process, they are oblivious to inter-server latencies which may have a huge impact on the maximum interaction path. *D-GMIN* works similar to *D-Nearest*. It differs in the way it connects incoming clients to one of the servers, i.e. the JOIN phase. Details of the JOIN phase are given in Algorithm-3. In *D-GMIN*, instead of exploiting only the minimum distance between a server and a client, we explicitly calculate the length of the interaction paths between the newly joined client and the clients that are already in the system by considering all server assignments. In other words, we consider assigning the new client to each of $|S|$ servers and find the maximum interaction path for each server assignment (Lines 4-12). Then, we select the server which ends up with the minimum of these maximum interaction paths as the assigned server of the new client (Line 10). If the capacity of that server is already reached, we select the server with the second least maximum interaction path and so on (Line 12). In *D-GMIN*, we also use the waiting list approach introduced in Section IV-B where re-assignment of the clients in the waiting lists is handled in a distributed manner.

In *D-GMIN* algorithm, we consider all $|S|$ number of possible server assignments and find the maximum interaction path between all client pairs for each server assignment, that is $|C| \times |C-1|/2$ many pairs in the worst case, which results

Algorithm 1 D-Nearest Algorithm

1: Start to wait $INITIAL_{message}$ from clients
2: **for each** received client c_i $INITIAL_{message}$ **do**
3: Reply back to client.
4: Receive client c_i server latency data pairs.
5: Start to wait REQ from system.
6: **for each** received REQ **do**
7: Extract operation REQ_{op}
8: **switch** (REQ_{op})
9: **case** HELLO**:**
10: **if** $SENDER_{req}$ is server **then**
11: Update client s_i latency value.
12: Update client s_i last interaction time.
13: **else**
14: Update client c_i latency value.
15: Update client c_i last interaction time.
16: **case** DELETE**:**
17: **if** $SENDER_{req}$ is server **then**
18: Apply dynamic leave part of algorithm
19: **case** JOIN**:**
20: Initialize S_{min} and $S_{optimal}$ to NULL.
21: Initialize $d(c_i, S_{min})$ to ∞.
22: **for each** server s_j in server list **do**
23: Use delay $d(c_i, s_j)$ between client c_i and server s_j
24: **if** $d(c_i, s_j) < d(c_i, S_{min})$ **then**
25: Set $S_{optimal}$ to s_j
26: Set $d(c_i, S_{min})$ to $d(c_i, s_j)$
27: **if** $Avail_{s_j} > 0$ **then**
28: Set S_{min} to s_j
29: **if** $S_{min} \neq S_{optimal}$ **then**
30: Assign client c_i to server S_{min}
31: Add c_i to $S_{optimal}$ to server waiting list $Wait_{s_{optimal}}$
32: Decrease the available capacity of server $Avail_{s_{optimal}}$
33: **else**
34: Assign client c_i to server $S_{optimal}$
35: Decrease the available capacity of server $Avail_{s_{optimal}}$
36: **end switch**

Algorithm 2 D-Nearest Algorithm Leave Phase

1: Server s_j detects client exit from system
2: Increment $Avail_{s_j}$ of server s_j
3: Control s_j waiting client list,$Wait_{s_j}$
4: **while** $Wait_{s_j} \neq$ NULL **do**
5: Remove first client c_{wait} from $Wait_{s_j}$
6: Send REQ delete operation to $S_{c_{wait}}$
7: Assign c_{wait} to server s_j
8: Decrement $Avail_{s_j}$ of server s_j

in $O(|S| \times |C|^2)$ run-time complexity.

Algorithm 3 D-GMIN JOIN Phase

```
Initialize S_min and S_optimal to NULL.
Initialize MAX_path to ∞.
Initialize temp_path
for each server s_j in server list do
    Assume c_i is assigned to s_j.
    Compute system wide maximum interaction path.
    Set temp_path to system wide maximum interaction path.
    if temp_path < MAX_path then
        if Avail_{s_j} > 0 then
            Set S_min to s_j
        Set MAX_path to temp_path
        Set S_optimal to s_j
if S_min ≠ S_optimal then
    Assign client c_i to server S_min
    Add c_i to S_optimal to server waiting list Wait_{s_optimal}
    Decrease the available capacity of server Avail_{s_optimal}
else
    Assign client c_i to server S_optimal
    Decrease the available capacity of server Avail_{s_optimal}
```

V. Experimental Setup

We have deployed a dynamic real-time distributed interactive application environment utilizing PlanetLab nodes around the world. We run our experiments with different number of clients and servers, more precisely, C and S values vary from 10 to 50, and 5 to 15, respectively. Client nodes are selected randomly from the assigned PlanetLab slice. Servers are not picked randomly but they are selected from different continents to construct a geographically distributed system.

The following experimental scenarios are considered for the system: (1) servers do not have any capacity constraint and (2) there exist different capacity constraints for each server ranging from 12 to 18. Additionally, in the former we have also run the system with different number of servers. In the latter, we have utilized different number of clients by fixing the number of servers to observe the effects of increasing clients in the system.

Experiments are run 30 minutes for each of the scenarios to construct a continuous system, and the reported results are average of 5 runs. Moreover, we randomly disconnect some of the clients and sleep them for a random time, and then reconnect to the system to handle (and also to observe the effects of) client failures.

VI. Results

There are two main performance metrics: maximum interaction path and average interaction delay. Minimizing the maximum interaction path is our main objective. Average interaction delay is defined as the average length of all the interaction paths between each pair of clients during the system execution.

In the first set of experiments, servers do not have any capacity constraint and can serve as many clients as needed. We have also considered different number of servers, and

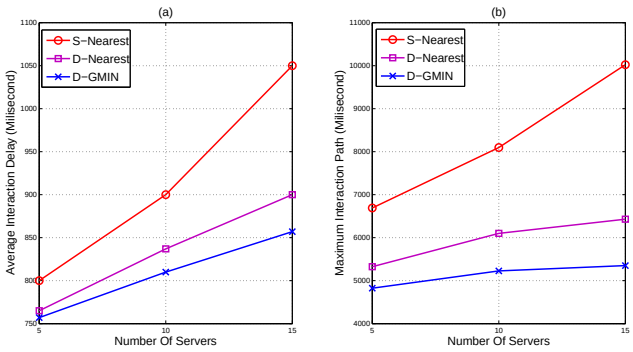

Fig. 2. Experimental results for different number of servers (a) Average Interaction Delay (b) Maximum Interaction Path

TABLE III
Performance improvement of our proposed distributed algorithms w.r.t the *S-Nearest* baseline

	D-Nearest		D-GMIN	
# of Servers	*Max*	*Avg*	*Max*	*Avg*
5	20.4%	4.4%	27.8%	5.4%
10	31.5%	7%	41.3%	10%
15	35.8%	14.2%	46.6%	18.4%

the first thing we have observed in results of Fig. 2 is *D-GMIN* algorithm outperforms both *D-Nearest* and *S-Nearest* algorithms in terms of maximum interaction path and average interaction delay. *S-Nearest* algorithm shows a fairly poor performance compared to other heuristics since it has no mechanism to handle the changing latency between nodes as shown in Fig. 1. In Fig. 2.a and b, intuitively, one would expect maximum interaction delay and average interaction path to decrease if the number of servers increases. In real world, where servers are superior machines with better performance compared to clients, that is true because inter-server latency would be less than the latency between clients and servers. However, our servers and clients are selected from a pool of PlanetLab nodes and these nodes have similar hardware and architectures. Therefore, when the number of servers increases it is more likely for clients to connect different servers which eventually increase the inter-server latencies, thus increase the maximum interaction path and average interaction delay.

In Fig. 2.b, in the best case, *D-GMIN* algorithm improves the performance of *S-Nearest* and *D-Nearest* algorithms by 46% and 17% respectively. In the worst case, it improves them by 28% and 10% respectively. Table.III summarizes relative performance improvement of our proposed *D-Nearest* and *D-GMIN* algorithms compared to the baseline *S-Nearest* algorithm with different number of servers and no capacity constraint (Max implies maximum interaction path and Avg implies average interaction delay).

In the second set of experiments, servers have limited capacity and it is not possible to connect to a particular server if its capacity is reached. Number of servers in Fig. 3 is fixed to 5. Different than what we have observed with unlimited servers,

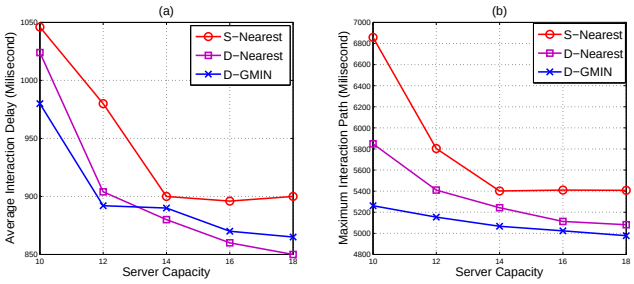

Fig. 3. Experimental results for different server capacities (a) Average Interaction Delay (b) Maximum Interaction Path

TABLE IV
PERFORMANCE IMPROVEMENT OF OUR PROPOSED DISTRIBUTED ALGORITHMS W.R.T THE *S-Nearest* BASELINE

	D−Nearest		D−GMIN	
Server Capacity	*Max*	*Avg*	*Max*	*Avg*
10	14.7%	2.1%	23.2%	6.3%
12	6.7%	7.7%	11.1%	8.9%
14	2.9%	2.2%	6.2%	1.1%
16	5.4%	4.0%	7.1%	2.9%
18	6.0%	5.5%	7.9%	3.8%

in this case, as seen in Fig. 3.a, with increasing capacity the *D-Nearest* approach seems to outperform *D-GMIN* in terms of average interaction delay. However, as observed in Fig. 3, when the server capacity increases, eventually *D-GMIN* performs better than *D-Nearest*. That is because *D-GMIN* has a better system wide awareness than *D-Nearest*, since it focuses on client to client latency rather than considering only client to server latency.

In Fig. 3.b, our *D-GMIN* heuristic improves the performance of baseline algorithm, *S-Nearest*, by 24% in the best case. Table.IV summarizes relative performance improvement of our proposed *D-Nearest* and *D-GMIN* algorithms compared to the baseline *S-Nearest* algorithm with varying server capacities.

Lastly, we fixed the number of servers to 5 with no server capacity and run our simulations with different number of clients from 10 to 50 to observe how the client number affects the performance of our proposed algorithms. Fig. 4 shows the dynamic behavior, that is the total number of client join and leave events in the system which increases with increasing number of clients as expected. Our proposed algorithms are able to handle this dynamic behavior in the system while improving the performance compared to the baseline algorithm. Since latency between any pair of node shows temporal variation unless there is no client with very poor connection compared to other nodes join the system, we expect maximum interaction path to be close to each other even if we increase the number of clients. As seen in Fig. 5.a and b, increasing number of clients tends to converge in terms of performance metrics that we evaluated, and *D-GMIN* outperforms the other algorithms.

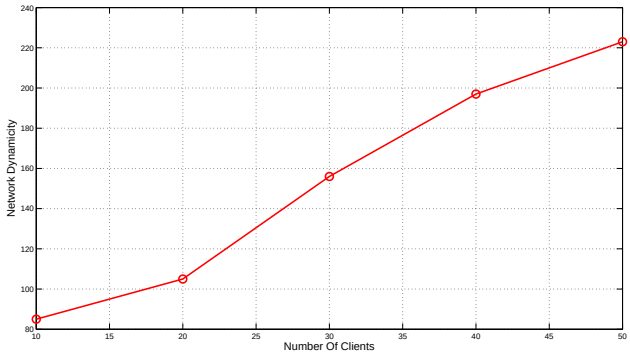

Fig. 4. Dynamicity in the network with different number of clients

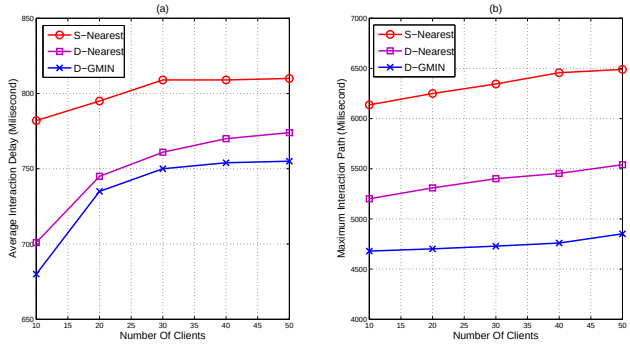

Fig. 5. Experimental results for different number of clients (a) Average Interaction Delay (b) Maximum Interaction Path

VII. CONCLUSION

In distributed interactive applications, each client is connected to one of the servers and pushes/retrieves updates in the system through their connected servers. Thus, any interaction between two clients consists of both client to server latency and inter-server latency which is called an interaction path. Our objective is to minimize the maximum of these interaction paths between any of the client pairs in the system. Previous works, that addressed the same problem, all considered a static system with previously calculated Internet latency values. Our work utilizes geographically distributed clients and servers interacting with each other on the PlanetLab platform in real-time, and we propose two distributed heuristic algorithms named *D-GMIN* and *D-Nearest* that improve the performance of the baseline algorithm, *S-Nearest*, up to 46%. As future work, we aim at improving our approach so that it can efficiently handle server failures. We also plan to integrate some filtering mechanisms into the system used for preventing waste of network and client resources by sharing updates not with all of the clients but just some of them.

VIII. ACKNOWLEDGEMENT

This work was partially supported by TUBITAK (The Scientific and Technical Research Council of Turkey) under

Grant 109M761.

REFERENCES

[1] T. Chang, G. Popescu, and C. Codella, "Scalable and efficient update dissemination for distributed interactive applications," in *Proceedings of the 22 nd International Conference on Distributed Computing Systems*, 2002.

[2] C. Jay, M. Glencross, and R. Hubbold, "Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment," *ACM Transactions on Computer-Human Interaction*, 2007.

[3] C. Nguyen, F. Safaei, and P. Boustead, "A distributed server architecture for providing immersive audio communication to massively multiplayer online games," in *Proceedings of the 12th IEEE International Conference on Networks*, 2004, pp. 170–176.

[4] L. D. Briceño, H. J. Siegel, A. A. Maciejewski, Y. Hong, B. Lock, M. N. Teli, F. Wedyan, C. Panaccione, C. Klumph, K. Willman, and C. Zhang, "Robust resource allocation in a massive multiplayer online gaming environment," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, 2009.

[5] L. Zhang and X. Tang, "Client assignment for improving interactivity in distributed interactive applications," in *The 30th IEEE International Conference on Computer Communications*, 2011.

[6] A. Myers, P. Dinda, and H. Zhang, "Performance characteristics of mirror servers on the internet," *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, 1999.

[7] S. Jamin, C. Jin, and A. R. Kurc, "Constrained mirror placement on the internet," in *IEEE Journal on Selected Areas in Communications*, 2001, pp. 31–40.

[8] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2001.

[9] L. Zhang and X. Tang, "The client assignment problem for continuous distributed interactive applications," in *31st International Conference on Distributed Computing Systems*, 2011.

[10] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: a lightweight network location service without virtual coordinates," in *Proceedings of the 2005 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2005, pp. 85–96.

[11] D. N. B. Ta and S. Zhou, "A network centric approach to enhancing the interactivity of large-scale distributed virtual environments," *Computer Communications*, 2006.

[12] M. S. D. Webb and D. S. Soh, "Adaptive client to mirrored-server assignment for massively multiplayer online games," *Multimedia Computing and Networking*, 2008.

[13] D. N. B. Ta, S. Zhou, and H. S. Shen, "Greedy algorithms for client assignment in large-scale distributed virtual environments," in *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 2006.

[14] H. Nishida and T. Nguyen, "Optimal client-server assignment for internet distributed systems," in *Proceedings of 20th International Conference on Computer Communications and Networks*, 2011.

[15] Y. Li and W. Cai, "Consistency-aware partitioning algorithm in multi-server distributed virtual environments," in *26th International Parallel and Distributed Processing Symposium*, 2012, pp. 798–807.

[16] S. Zhou, W. Cai, B.-S. Lee, and S. J. Turner, "Time-space consistency in large-scale distributed virtual environments," in *ACM Transactions on Modelling and Computer Simulation*, 2004, pp. 31–47.

[17] X. Tang and S. Zhou, "Update scheduling for improving consistency in distributed virtual environments," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 765–777, 2010.

[18] L. Zhang and X. Tang, "Optimizing client assignment for enhancing interactivity in distributed interactive applications," in *Networking, IEEE/ACM Transactions on*, vol. 20, no. 6, 2012, pp. 1707–1720.

[19] ——, "The client assignment problem for continuous distributed interactive applications: Analysis, algorithms, and evaluation," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, 2013.

[20] Planetlab-all-pairs-pings data set. [Online]. Available: http://pdos.lcs.mit.edu/~strib/

[21] Lightweight approach to network positioning. [Online]. Available: http://www.cs.cornell.edu/People/egs/meridian/

[22] MIT latency data set, king. [Online]. Available: http://pdos.csail.mit.edu/p2psim/kingdata/