Characterization of CAR Servers for Augmented Reality Marker Tracking

Víctor Fernández, Juan M. Orduña, Pedro Morillo, Vicente Cavero [1]

Abstract— **Collaborative Augmented Reality (CAR) systems allow multiple users to share a real world environment including computer-generated images in real time. Currently, the hardware features of most mobile phones provide excellent multimedia services, and it also includes wireless network capabilities that offer a natural platform for CAR systems. However, the performance of the mobile CAR applications under different conditions, like the number and type of devices in the system, has not been studied yet.**

This paper presents the experimental characterization of CAR systems based on mobile phones, providing quantitative results about well-known performance metrics in distributed systems like system throughput and system response times. The characterization results show that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server, although it is not a fixed value and it is inversely related to the number of processor cores. Also, the results show that throughput of CAR systems heavily depends on the kind of client devices, but CAR systems can efficiently support some hundreds of clients in any case. Another important result is that the CAR system throughput is limited by the server I/O in some cases. Therefore, any improvement in CAR systems should be addressed to alleviate the server I/O, even though it may add computational overhead to the server.

Key words— **Collaborative Augmented Reality; Mobile Phones; Performance evaluation**

I. Introduction

Augmented Reality (AR) systems have been widely used in numerous applications such as medical procedures, scientific visualization, manufacturing automation, cultural heritage and military applications [1]. The term Augmented Reality (AR) defines computer graphic procedures or applications where the real-world view is superimposed by computer-generated objects in real-time[2]. From the beginning of AR systems, the potential of collaborative AR (CAR) systems was exploited for different activities such as Collaborative Computing [3] or Teleconferencing [4]. Wearable devices were used to provide CAR systems, where a wearable AR user could collaborate with a remote user at a desktop computer [5].

On other hand, lot of devices comprising a computing embedded system pervade our daily life, and they have been used for CAR systems. One of these devices are mobile phones [6]. Effectively, current mobile phones have full color displays, integrated cameras, fast processors and even dedicated 3D graphics chips, and they have become an ideal platform for CAR systems [7]. However, the wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems, can have significant effects on the performance of the CAR application, in terms of system latency, frames per second or number of supported clients with certain latency levels. Taking into account that CAR applications should be interactive, the design of an efficient CAR application must take into account these effects in order to fulfill the required specifications. Therefore, in order to design efficient CAR systems based on mobile devices, it is necessary to characterize the behavior of these systems in terms of average response times under different conditions, as well as system throughput. In turn, this characterization should be performed on both the client devices (mobile phones) and the system server(s).

In a previous work, we performed a performance characterization of different mobile phones for Collaborative Augmented Reality applications, [8]. In order to achieve this goal, we implemented a simple CAR application on a real system and measured the performance achieved with different mobile phones. The results showed that the most time consuming stage in a CAR application is the marker detection stage, followed by the image acquisition stage, the rendering stage and finally, the transmission stage. In this paper, we present a performance characterization from the server side, measuring the system response time and system throughput when varying different systems parameters like the number of clients in the system, the Area of Interest (AOI) [9] of clients (i.e., the number of neighbors to which messages should be sent), and the cycle time of clients. For characterization purposes, we have developed a multithreaded CAR server that supports simulated clients (simulated mobile devices) with the behavior measured in our previous work [8]. We have timestamped every message generated within this CAR system, in order to measure the performance of every device. The characterization results show that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server. Although the CPU threshold is not a fixed value, it is inversely related to the number of processor cores. Moreover, the results show that the CAR systems throughput heavily depends on the kind of client devices, but for certain kind of devices, the system bottleneck is the server I/O. These results suggest that any improvement in CAR systems should be addressed to alleviate the server I/O, even though it may add computational overhead to the server.

The rest of the paper is organized as follows: Sec-

[1]Departamento de Informática, Universidad de Valencia, e-mail: {Victor.Fernandez-Bauset, Juan.Orduna, Pedro.Morillo, Vicente.Cavero}@uv.es

tion II shows some related work about CAR applications on mobile phones. Section III describes the characterization setup, and Section IV shows the characterization results. Finally, Section V presents some conclusion remarks and the future work to be done.

II. Related Work

Augmented Reality superimposes multimedia content - 3D object, text, sound, etc - to real world through a display or screen. In order to locate digital contents on a specific image of the real world point, some references within the image is needed. These references are known as markers, and two methods are usually used: natural feature tracking and fiducial marker tracking. The former method uses interest point detectors and matching schemes to associate 2D locations on the video with 3D locations [10]. This process can be grouped in three big phases: interest point detection, creation of descriptor vectors for these interest points, and comparison of vectors with the database [11]. The latter method uses fiducial markers to find a specific position of real world. This process can be divided in three phases: edge detection, rejection of quadrangles that are too large or too small, and checking against the set of known patterns [10].

Although real-time natural feature tracking over mobile devices has been currently achieved [10], fiducial marker tracking is more widely used, because it allows simultaneous computational robustness and efficiency. A large number of locations and objects can be efficiently labeled by encoding unique identifiers on the markers. Additionally, the markers can be detected with angles near to 90 degrees [10].

III. Characterization setup

In order to analyze the behavior of CAR systems based on mobile devices, we have developed a multithreaded CAR server that supports simulated clients (simulated mobile devices) with the behavior measured in our previous work [8]. We have timestamped every message generated within this CAR system, in order to measure the performance of every device. The system configuration will consist of one server, and a certain amount of mobile devices that are scanning the visual space of their video camera looking for a marker that will be converted into a 3D object in their display. The main performance metrics in distributed systems are throughput and latency [12]. However, in order to avoid clock skews when measuring the system latency in distributed systems, the same device should measure the initial and final time. Therefore, we consider round-trip times instead of system latencies.

Since we are considering collaborative systems, after each updating of the object location, the mobile device will send a location update message (containing the new location) to each of its neighbor devices (defined by the AOI size) through the server (that is, it sends the location update message to the server, and then the server re-sends the message to the appropriate clients). For characterization purposes, the destination clients return an acknowledgment message (ACK) to the server, which, in turn, forwards it to the source client. When the source client has received the ACK messages corresponding to the location update from all the clients in its AOI, then it computes the average system response for that location update. Figure 1 illustrates the action cycle that takes place for each of the mobile clients in the system.

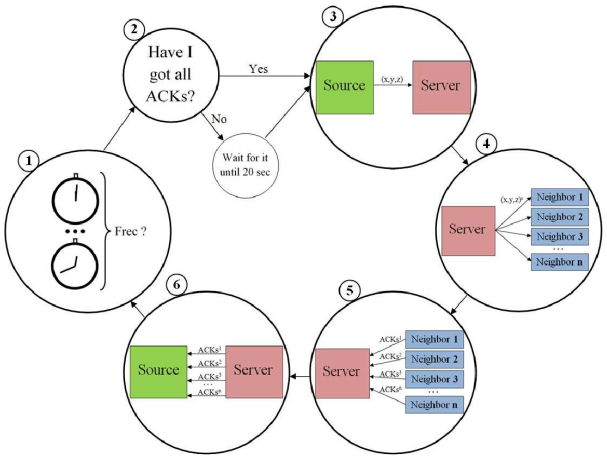

Fig. 1. Stages of the action cycle in each mobile device.

Once the message with the location update is sent, the action cycle performed by each client is composed of the following steps: first, it performs one new image acquisition followed by a marker detection stages. Then, the client waits until the cycle period (determined by the action frequency, a system parameter) finishes. Next, if the acknowledgments from all the neighbors have been received, a new message with the new marker location is sent. If not all the acknowledgments have been received, then it waits until a maximum threshold of 20 seconds, and then a new round of messages (with the latest marker location) are sent to the neighbors through the server. The neighbors simply returns an ACK message to the sender device through the server. The server simply forwards the messages to the corresponding destination clients. It must be noticed that the mobile devices will not send a new round of messages with a new location update until it has received the acknowledgment message from all its neighbors, even although new marker detection stages have been completed in the device.

This characterization setup considers that all the required static content in the scene has been loaded. According to recent works [13], in these cases the network bandwidth required is less than 50 kbps for performing this information exchange. Since we are using a Gigabit Ethernet, we ensure that network bandwidth does not become a system bottleneck.

The system latency provided for each location update is computed by recording a timestamp when the first message is sent to the server. Next, a second

timestamp is recorded with the last ACK message for that location update received from the server. The system response time is computed by subtracting these two timestamps. The server response time is computed by timestamping both each message forwarded from each client and the reception of the corresponding ACK message from the destination client. Also, the percentage of CPU utilization is measured both in the server and the mobile devices every half second.

We have implemented a multithreaded server, where each server thread manages a group of clients within a given AOI. Thus, for example, with a system configuration of 500 mobile clients and an AOI size of 10 clients, we have 50 server threads, each thread supporting 10 clients. We have considered a maximum configuration of 1000 clients, resulting in 100 server threads. For system characterization purposes, we have considered a single server. Nevertheless, the system performance greatly depends on the server implementation (see the results in Section IV). Therefore, in order to make a robust characterization of CAR systems, we have implemented two different versions of the server, denoted as *passive* and *active* server. The *active* server forwards each location update message to the corresponding neighbor clients, and it collects the ACK messages received from the neighbor clients. When all the ACK messages corresponding to a location update message have been received, then a single ACK message is sent to the corresponding source client. The *passive* server simply forwards each location update message to the corresponding neighbor clients and the ACK messages received from the neighbor clients to the source client. In this sense, no computations neither data structures are needed in the server, although more messages are exchanged between the server and the clients (the network traffic increases).

On other hand, each client process simulates 50 mobile devices, using two threads per simulated device. We have uniformly distributed the number of the required client processes for each system configuration. Since we have 10 desktop computers available for hosting the clients, the configuration for 1000 clients consists of 10 computers hosting 5 client processes each (100 threads per computer hosting clients).

The previous work showed that Google phone HTC Nexus One was the fastest device, with a period cycle of 167.11 milliseconds, while the Motorola Milestone was the slowest one, with a period cycle of 698.34 milliseconds. We have considered these values as the limits for characterization purposes. Also, we have considered four different values for the AOI size: 5, 10, 20, and 25 neighbor clients. Finally, we have considered a number of clients in the system ranging from 100 to 1000. It must be noticed that usually, actual CAR applications do not contain more than a hundred clients (for example, more than a hundred persons within the same lounge using collaborative Augmented Reality for studying art masterpieces), due to the size of the augmented models. Thus, reaching thousands of clients clearly exceed the worst case for this kind of applications.

TABLE I
System performance for the active server implementation with an AOI of 10 neighbors

	a-One		a-Miles	
AOI 10	RT	CPU	RT	CPU
100	70.27	10.9	72.89	7.1
200	68.85	21.8	69.35	13.0
300	73.27	36.4	73.67	17.8
400	74.81	44.0	78.13	18.8
500	79.71	57.0	75.9	26.0
600	83.58	70.0	80.35	31.7
700	83.85	82.3	81.73	31.0
800	179.75	84.2	78.69	40.8
900	222.94	84.0	76.32	43.9
1000	255.67	85.2	78.29	45.1

IV. Performance Evaluation

We have measured the average system response time for all the location updates send by all the clients in the system. In this sense, we have considered the system response time (in milliseconds) for each location update as the time required for receiving all the acknowledgments from the neighbor clients in the AOI for each update sent by a client. Also, we have measured the average response time in the server (in milliseconds) as the time required by the destination clients to answer the server messages. Additionally, we have computed the standard deviation for the response times, and the percentage of the CPU utilization in the system server, since it can easily become the system bottleneck. The computer platform hosting the system server is a Intel Core 2 Duo E8400 CPU running at 3.00 GHz with 4 Gbytes of RAM, executing an Ubuntu Linux distribution with the 3.0.0-14-generic x86_64 operating system kernel.

Table I shows the performance evaluation results for the active server implementation (denoted with the "a-" prefix in the tables) and considering an AOI size of 10 neighbors for each client. The most-left column in this table shows the number of clients in the system (and the number of neighbor clients in the AOI). The values in this columns range from 100 to 1000 clients in the system. Next, there are two groups of three columns each that shows the results for the fastest (HTC Nexus One, whose results are labeled as "One") and the slowest client device considered (Motorola Milestone, whose results are labeled as "Miles"). The first two columns in each group of columns show the average system response time (in milliseconds) and its standard deviation. The third column shows the percentage of the CPU utilization in the system server.

Table I shows clearly different behavior and performance for different client devices. Thus, comparing

TABLE II
System performance for the passive server implementation with an AOI of 10 neighbors

	p-One		p-Miles	
AOI 10	RT	CPU	RT	CPU
100	78.23	16.0	77.03	9.9
200	74.02	25.2	78.16	26.0
300	79.45	37.0	83.6	25.0
400	81.02	59.6	82.54	23.8
500	93.61	68.7	88.27	31.3
600	147.73	83.8	87.66	33.0
700	195.47	83.2	88.45	33.7
800	237.71	85.9	94.43	62.0
900	270.25	84.2	84.8	69.3
1000	300.96	84.2	87.72	46.5

TABLE III
Server Response Times

	p-One			
AOI 10	RT	CPU	SRT	MaxSRT
100	78,23	16	18,2	20,15
200	74,02	25,2	18,5	24,9
300	79,45	37	25,78	28,62
400	81,02	59,6	27,71	30,4
500	93,61	68,7	31,83	40,5
600	147,73	83,8	40,93	44,37
700	195,47	83,2	49,85	52,39
800	237,71	85,9	58,08	62,23
900	270,25	84,2	64,79	68,9
1000	300,96	84,2	68,77	75,48

the system response time (column RT) for the a-One (i.e., active server implementation and all clients using a Nexus One device) and a-Miles configurations, it can be seen that the RT slightly increases (or decreases) as the number of clients in the system increases in the case of the a-Miles, ranging from 72.89 milliseconds to 81.73 milliseconds. However, in the case of the a-One, when the population reaches 800 clients there is a huge increase in both the RT value and in its standard deviation with respect to the values for smaller population sizes. For higher population sizes, the increase in the RT values is also significant.

Table II shows the performance results for a CAR system with the same configuration except that the server implementation is now the passive server (denoted with a "p-" prefix). This table shows a similar behavior of the CAR system to the one shown in Table I for the active server. The only difference is the huge increase in the RT values of the "p-One" for a smaller population size (600 clients instead of 800 clients in the "a-One" configuration). These results show that the system reaches saturation for a population size of 800 clients in the case of a-One, and for a population size of 600 clients in the case of the p-One. However, in both cases the saturation starts when the percentage of CPU utilization in the system server reaches around 84% (83,8% in the passive server and 84,2% in the active server implementation). The configurations "a-Miles" and "p-Miles" do not reach saturation, since the RT remains almost constant as the population size increases. The reason for the different system behavior, when changing the devices, is the greater period cycle of the Motorola Milestone, which imposes a lower system workload since it generates new location updates at a much lower rate.

Regarding the acceptable limits for the RT values, some works consider 250 milliseconds as the response time limit for providing interactivity to human users [14]. If we consider such threshold value, then we can state that the throughput limit with the HTC Nexus One is 900 clients when using the active server, and 800 when using the passive server. That is, the system performance is surprisingly improved when the server reduces the traffic exchanged with the source clients (it collects all the ACK messages and only sends the source client a single ACK) at the cost of performing more computations (counting the ACK received for each update message). In order to find the reason for this behavior, we have measured the server response times for all the configurations considered. Table III shows the same results for the "p-one" configuration shown in table II, but now adding the corresponding average response times in the server (the average response time required by clients for sending back their ACK messages to the server, measured from the server) and their maximum values.

Table III shows that the average response times do not reach 70 milliseconds for the p-One device regardless of the population size, while the maximum values for the server response times do not reach 80 milliseconds. These values indicate that the system bottleneck (the reason for the significant average system response times shown in Tables II and I) is located in the server. Since the comparison between these two tables shows that the active server achieves better performance than the passive server, these results suggest that the server I/O with the source client is the system bottleneck. It is important to notice that each source client in the passive server must not only receive all the ACKs for each message sent, but also it must send an ACK for every location update received from its neighbors. Although the active server adds more computational workload to the server for the same system workload than the passive server, it alleviates the server I/O with each client, and therefore the performance is improved.

In order to show that the behaviors shown in Tables I and II are consistent for other workloads, Tables IV and V show the results corresponding to a system configuration when all the clients have an AOI size of 20 neighbor clients. These tables show behaviors that are very similar to the ones shown in Tables I and II, except for the fact that the RT values are in general higher, and the system reaches

TABLE IV
System performance for the active server implementation with an AOI of 20 neighbors

	a-One		a-Miles	
AOI 20	RT	CPU	RT	CPU
100	77.37	23.8	78.65	12.8
200	77.13	38.0	77.69	16.0
300	79.59	56.6	84.26	35.4
400	83.07	74.2	96.71	38.0
500	195.92	82.9	81.19	40.0
600	246.46	82.8	103.62	46.0
700	297.3	82.8	97.92	47.0
800	350.88	84.2	82.63	62.4
900	394.66	86.0	93.31	56.5
1000	410.14	87.0	88.52	66.3

TABLE V
System performance for the passive server implementation with an AOI of 20 neighbors

	p-One		p-Miles	
AOI 20	RT	CPU	RT	CPU
100	88.22	27.6	93.88	20.2
200	94.57	45.5	94.98	21.2
300	128.16	73.3	98.01	29.7
400	195.92	84.2	129.24	37.7
500	273.58	84.0	101.96	43.0
600	326.79	86.2	121.53	50.5
700	400.45	87.0	148.84	83.9
800	447.02	87.0	110.41	65.3
900	473.99	86.0	101.4	67.3
1000	467.04	86.9	106.66	74.7

saturation for lower population sizes. The results for the Milestone device show that the system does not reach saturation (there is not a significant jump in the RT values as the population size increases) in any of the tables. However, the results shown in Table IV for the a-One show a jump in the RT values for a population size of 500 clients, when the percentage of CPU utilization is 82.9%. In the case of the p-One configuration (Table V), the disruption occurs for a population size of 400 clients, when the percentage of CPU utilization is 84.2%. That is, again the system saturation point is a percentage of CPU utilization in the system server of around 84%.

Finally, we have studied the saturation point of the CAR configurations considered. Although they are not shown here for the sake of shortness, we have obtained a percentage of CPU utilization in the server around 84% as the system saturation point for all the configurations considered. However, similar distributed systems like Distributed Virtual Environments (DVEs) were reported to reach saturation when any of the server reaches around 95-98% of CPU utilization [15]. The difference between 85% and 98% of CPU utilization for reaching the saturation point can be explained by the shared memory architecture of current multicore processors (the dual core processor in the computer platform used as simulation server). The synchronization of the kernel calls, together with the synchronization among threads in the application, prevent the CAR system from fully exploiting the computational power of all the processing cores at the same time, reaching saturation for a lower overall percentage of CPU utilization. The performance characterization of DVEs was performed on single core processors, and therefore those systems reached around 98% before reaching saturation. In order to show that this is the reason for this behavior, we have performed the same tests not only with the computer platform described above (dual core processor), but also with a different multicore computer platform, consisting of an Intel(R) Core(TM) i7 960 CPU (eight cores) running at 3.20GHz with 8 Gbytes of RAM and executing a SuSE Linux 2.6.37.6-0.5-desktop operating system. A representative example of the results obtained in such tests is shown in Table VI, that shows the results for the p-One configuration with an AOI size of 25 neighbor clients.

TABLE VI
Performance evaluation for two different computer platforms

	8 cores (i7)		2 cores (Core Duo)	
AOI 25	RT	CPU	RT	CPU
100	90.91	24.1	96.03	60.4
300	98.65	33.6	167.58	86.1
500	228.98	43.5	357.13	92
700	357.95	45.3	480.87	85.3
900	478.95	48.3	505.89	89.0

Table VI shows that the average system response time hugely increases when the population size is composed of 500 clients for the case of the Intel i7 platform, passing from 98.65 milliseconds to 228.98 milliseconds, while the standard deviation for this parameter also double its value. That is, the system saturation point for this server platform is a population size of 500 clients, and for this saturation point the overall percentage of CPU utilization for this execution is 42.46% (fourth most-left column in table VI). However, the results obtained with an Intel Dual Core processor show that the system reaches saturation with 300 clients, where the average system response time jumps from 96.03 milliseconds to 167.58 milliseconds.For this system response time, the overall percentage of CPU utilization is 86.1%. That is, as it could be expected, the system throughput is increased when a more powerful processor is used as the system server, but the saturation point of the CAR system is not a fixed value for the overall percentage of the CPU utilization, and it depends on the number of processor cores. Although they are not here due to space limitations, the system saturation point around 43% of CPU utilization arose for all the configurations tested with this platform.

V. Conclusions and Future work

This paper has proposed the experimental characterization of CAR systems based on mobile phones, providing quantitative results about well-known performance metrics in distributed systems like system throughput and system response times. The results show that the system saturation point depends on the overall percentage of CPU utilization in the computer platform acting as the system server, although it is not a fixed value and it is inversely related to the number of processor cores. Also, the results show that throughput of CAR systems heavily depends on the kind of client devices, but CAR systems can efficiently support some hundreds of clients in any case. Another important result is that the CAR system throughput is limited by the server I/O. Therefore, any improvement in CAR systems should be addressed to alleviate the server I/O, even though it may add computational overhead to the server.

As a future work to be done, we plan to evaluate different CAR systems implementations oriented to minimize the server I/O workload.

Acknowledgment

This work has been jointly supported by the Spanish MICINN and the European Commission FEDER funds, under grants Consolider-Ingenio 2010 CSD2006-00046 and TIN2009-14475-C04.

References

[1] S. Cawood and M. Fiala, *Augmented Reality: A Practical Guide*, Pragmatic Bookshelf, 2008.

[2] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *Computer Graphics and Applications, IEEE*, vol. 21, no. 6, pp. 34 –47, 2001.

[3] M. Billinghurst, I. Poupyrev, H. Kato, and R. May, "Mixing realities in shared space: an augmented reality interface for collaborative computing," in *IEEE International Conference on Multimedia and Expo (ICME 2000)*, 2000, vol. 3, pp. 1641–1644.

[4] M. Billinghurst and H. Kato, "Real world teleconferencing," in *Proc. of the conference on Human Factors in Computing Systems (CHI 99)*, 1999.

[5] Wayne Piekarski and Bruce H. Thomas, "Tinmith-hand: Unified user interface technology for mobile outdoor augmented reality and indoor virtual reality," 2002.

[6] Mathias Mahring, Christian Lessig, and Oliver Bimber, "Video see-through ar on consumer cell-phones.," in *ISMAR'04*, 2004, pp. 252–253.

[7] A. Henrysson, M. Billinghurst, and M. Ollila, "Face to face collaborative ar on mobile phones," in *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, October 2005, pp. 80 – 89.

[8] Victor Fernández Bauset, Juan M. Orduña, and Pedro Morillo, "Performance characterization on mobile phones for collaborative augmented reality (car) applications," in *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, Washington, DC, USA, 2011, DS-RT '11, pp. 52–53, IEEE Computer Society.

[9] S. Singhal and M. Zyda, *Networked Virtual Environments*, ACM Press, 1999.

[10] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg, "Pose tracking from natural features on mobile phones," in *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Washington, DC, USA, 2008, ISMAR '08, pp. 125–134, IEEE Computer Society.

[11] Seung Eun Lee, Yong Zhang, Zhen Fang, S. Srinivasan, R. Iyer, and D. Newell, "Accelerating mobile augmented reality on a handheld platform," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, October 2009, pp. 419 –426.

[12] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, 1997.

[13] Tuomas Kantonen, "Augmented collaboration in mixed environments," M.S. thesis, Helsinky University of Technology, 2009.

[14] T. Henderson and S. Bhatti, "Networked games: a qos-sensitive application for qos-insensitive users?," in *Proceedings of the ACM SIGCOMM 2003.* 2003, pp. 141–147, ACM Press / ACM SIGCOMM.

[15] P. Morillo, J. M. Orduna, M. Fernández, and J. Duato, "Improving the performance of distributed virtual environment systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 7, pp. 637–649, 2005.