Fast Delivery of Game Events with an Optimistic Synchronization Mechanism in Massive Multiplayer Online Games

Stefano Ferretti
Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7, 40127
Bologna, Italy
sferrett@cs.unibo.it

Marco Roccetti
Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7, 40127
Bologna, Italy
roccetti@cs.unibo.it

ABSTRACT

As smart players often win MMOG sessions by adopting frantic gaming strategies along the game evolution, also the system activities concerned with the distributed support of MMOGs must advance at a very fast pace. Unfortunately, MMOGs' responsiveness requirements are hardly met when pessimistic approaches are adopted to synchronize the game event exchange activities among game servers. In this paper we show how MMOGs are better supported by optimistic synchronization schemes coupled with mechanisms that exploit the semantics of games. Results obtained from an experimental assessment of our developed scheme demonstrate the validity of our claim.

Categories and Subject Descriptors

K.8.0 [**Computing Milieux**]: PERSONAL COMPUTING—*Games*

General Terms

Algorithms, Synchronization, Performance, Evaluation

Keywords

Massive Multiplayer Online Games, Online Entertainment, Optimistic Synchronization, Responsiveness

1. INTRODUCTION

Massively Multiplayer Online Games (MMOGs) require considerable investments and efforts to manage the computing and network infrastructure needed to guarantee compelling gaming experiences. While the costs of hardware technologies decrease, the amount of users that connect to the Internet through a variety of different network-enabled terminals increase. This potential swarm of players impose the use of highly reliable, responsive software solutions able to hide latencies and overheads that may affect the usability of games over the Internet.

In this sense, an interesting proposal to support MMOGs amounts to exploiting a constellation of *Mirrored Game State Servers* (*GSSs*) which are geographically dispersed over the net [5, 11, 12, 13, 14, 24, 25]. Each *GSS* maintains a local, replicated representation of the state of the game. Based on this approach, each player connects to its *nearest GSS* and communicates with it in a classic client/server style [5]. In turn, each *GSS* is interconnected with all other *GSSs* resembling a P2P architecture [7, 18]. With each new action performed by players connected to a given *GSS*, the *GSS* collects the corresponding event, notifies it to other *GSSs*, updates the game state locally and, finally, communicates the newly computed game state to their connected players.

This architectural solution clearly facilitates the development of MMOGs where a potentially high number of players connect and play within the same game session. Indeed, according to this solution only subsets of users connect to the same *GSS* (and different users may be connected to different *GSSs* while participating to the same game session), thus reducing network and computational overheads at the servers-side. Then, a high level of fault-tolerance is ensured, as no single point of failure exists within the architecture. Needless to say, an efficient event synchronization scheme is to be employed to guarantee a consistent and responsive evolution of the game state among all *GSSs*.

In this scenario, a recent study was proposed where the goal of uplifting the playability degree of MMOGs is achieved by maintaining the game event delivery delays under a human perceptivity threshold, whilst preserving the game state consistency [13]. At the basis of this approach lies the idea of exploiting the notion of *obsolescence*. In essence, obsolescence allows each single *GSS* to autonomously drop those game events that lose their importance during the game evolution so as to speed up the event notification and processing activities to gain interactivity while preserving some form of state consistency.

Obviously, not any event in a game may be considered as

obsolete. Alongside the notion of obsolescence, in fact, also the notion of *correlation* among events has to be taken into account. Indeed, events may exist that are correlated with each other. To simply explain the notion of correlation, think of an event which cannot be considered as obsolete as further events may come into the picture that correlate it to the final game state [13].

Another interesting effect of correlation amounts to the fact that only correlated events are required to be processed in the same order at all the *GSSs* to maintain the consistency of the game state. On the other hand, different delivery orders are possible when a given sequence of non-correlated events has to be delivered to different *GSSs*.

Based on the notions of obsolescence and correlation, several schemes have been recently introduced that exploit these two properties for an augmented interactivity [24]. Of these, a first reactive approach has been proposed that it is able to measure the interactivity degree provided by the system. In essence, as long as such an interactivity degree rests equal to an acceptable value, then normal delivery operations are performed among *GSSs*. Instead, as soon as the measured interactivity degree decreases below a given game perceptivity threshold at a given *GSS*, a procedure is activated at that *GSS* which skips the processing of obsolete game events so as to report the interactivity level to an acceptable value (Interactivity Restoring).

Alternatively, another strategy exists that aims at proactively avoiding any loss of interactivity before it happens. In essence, obsolete events are discarded in advance based on a given dropping probability which depends on the measured values of the interactivity degree (Interactivity-Loss Avoidance).

The problem with all these strategies amounts to the fact that they have been developed based on a conservative approach. Technically speaking, the decision at a given *GSS* whether an event e can be processed is delayed till the point when all non-obsolete events preceding e are received by that *GSS*. (Hereinafter, we refer to those synchronization mechanisms as COS, *Conservative Obsolescence-based Synchronization*, schemes.)

In this paper, we present a novel game event synchronization mechanism for mirrored game servers that exploits the notions of obsolescence and correlation based on an optimistic synchronization approach. We refer to our mechanism as OOS, *Optimistic Obsolescence-based Synchronization* scheme.

Based on our OOS mechanism, with each new game event e received at a given *GSS*, an immediate check for detecting the obsolescence of e is carried out. If this check succeeds, the event is immediately discarded. Otherwise, another check is performed to verify whether later events e^* exist that are correlated to e and have been already processed. In this case, a rollback procedure is performed which is as follows. All those events e^* are rolled back and a new game state is computed that can be now safely delivered to all the connected players. Obviously, during the rollback procedure some of these events e^* may exist that have become obsolete as long as new values of the game state variables are recomputed. In such a case, those obsolete events are simply dropped.

It is widely accepted that the main advantage of optimistic strategies, w.r.t. conservative, amounts to the fact that they perform better in fast paced games where a continuous rate of the game advancement is crucial [5]. The experimental study we have conducted confirms this consideration, especially when the notion of obsolescence and correlation are exploited to gain interactivity.

As a final remark, we wish to point out that our mechanism has been also recently embedded into a MMOG we developed to allow customers to play a car racing game (inspired to Armagetron) over the Web [3]. Usage of this game with many players has confirmed that resorting to our OOS scheme greatly improves the playability degree of online games across the Internet.

The remainder of this paper is organized as follows. First, we provide the reader with an overview our system model (Section 2). Then, we present our novel scheme along with a discussion on experimental results obtained from a simulative assessment (Section 3). A comparison of our scheme with other ones proposed in the literature is reported in Section 4. Finally, we conclude the paper with some closing comments (Section 5).

2. THE MODEL: A SUMMARY

This Section is devoted to present the model at the basis of our approach. Thus, we begin with a discussion on issues concerned with the need for interactivity in a MMOG, followed by an overview of the notions of obsolescence and correlation.

2.1 MMOGs and Interactivity

It is well known that over a best-effort network real-time performance is often an illusive objective to achieve. Based on scientific literature, to assess the level of responsiveness provided by the system, a human perceptivity threshold may be defined that represents the limit above which the interaction among players is not satisfied. We term it *Game Interaction Threshold* (*GIT*). Typical values are in the range of 100-200 ms, as discussed at length in [1, 2, 9, 26].

Denoting with $T_g^p(e)$ the time of generation of a given event e at a given *GSS* p, and with $T_d^q(e)$ the delivery time of e at a given *GSS* q, we introduce a metric that measures the *Game Time Difference* between the time of generation of an event e and its delivery among communicating *GSSs*, that is $GTD_{p,q}(e) = T_d^q(e) - T_g^p(e)$. This value is an estimation of the interactivity degree provided by the system; thus, interactive game applications are well supported only if the *GTDs* of the generated game events are kept below the limit provided by *GIT* at all *GSSs*.

Obviously, our approach rests upon the assumption that a global notion of time is available across all *GSSs*. This result may be accomplished based on a variety of different approaches, ranging from the use of physical clocks' software synchronization schemes to the use of technological

synchronization devices, such as GPS for example [4, 8, 16, 23].

Based on a global notion of time, we assume that it is possible to provide a total order on the set of all the game events generated during the game evolution. We term this as correct *timestamp order*, meaning that a correct evolution of the game may be ensured at each *GSS* if events are processed according to this timestamp order. Unfortunately, a major concern is that large computational and communication overheads are to be paid if traditional event synchronization schemes are adopted to enforce a timestamp order on the game event processing activity at each *GSS* [5, 7, 15, 19, 20, 21].

2.2 Correlation and Obsolescence

To overcome the problem mentioned above, the notion of *correlation* has been recently introduced [13]. Correlation among game events may be characterized as a property which states that two game events e_i, e_j are correlated if different orders of execution of the two game events lead to different game states. Examples of correlated game events typically involve game events, possibly generated by different players, that act on the same game elements. Independent movements of different virtual characters, instead, are examples of non-correlated game events.

Hence, to provide players with a uniform evolution of the game, it is enough that correlated game events are processed by all *GSSs* respecting their correct timestamp order. Instead, no ordering guarantee is needed to process non-correlated events, as their delivery in different orders at different *GSSs* do not alter the game state. Such correlation-based order has the main advantage of reducing the synchronization overheads.

Another important concept exploited in our model amounts to the notion of obsolescence. That is, given two subsequent game events e_i, e_j $(T_g^p(e_j) > T_g^p(e_i))$ it may be the case when processing e_j without e_i leads to the same final state that would be reached if both events were processed in the correct order (i.e., e_i becomes obsolete). Recent studies demonstrated that by exploiting the semantics of the game, there exist many situations where fresher game events annul the importance of previous events. For example, knowing the position of a character at a given time may be no longer important after a certain time period, if the position of the character has changed. It is also worth noticing that the notion of obsolescence cannot be applied to e_j and e_i when other events correlated to e_i have been generated within the time interval $[T_g^p(e_i), T_g^p(e_j)]$. These game events may alter the evolution of the plot thus making unapplicable the notion of obsolescence. Deeper details about obsolescence and correlation may be found in [13].

3. AN OOS SCHEME WITH EXPERIMENTS

In this Section we present the OOS scheme we have devised followed by a discussion on results obtained from an evaluation assessment we conducted.

3.1 The Scheme

We developed a novel *Optimistic Obsolescence-based Synchronization* (OOS) scheme that guarantees a correlation-based order delivery strategy only for those events that are not discarded due to obsolescence. Our OOS algorithm is reported in Figure 1, where all the actions accomplished by a given *GSS* when a new game event is received are reported.

Specifically, according to our OOS scheme, each *GSS* verifies if a given event e may be already identified as obsolete (line 2). In this case, e is dropped (line 3). Otherwise, a check is carried out to control whether any game events e_i, correlated to e and generated after e, have been already processed (lines 5-6). If this check succeeds, then a rollback procedure is performed (which is based on a standard incremental state saving technique [15]) where all these events e_i are rolled back (line 7). At this point, e is processed (line 8), followed by the execution of all those rolled back events which are not obsolete (lines 9-12). In fact, obsolete events are discarded during the rollback (lines 10-11). If the check fails that determines that a rollback procedure is needed, e is directly processed (line 14).

It is easy to observe that our OOS scheme respects the correlation-based order and guarantees that only useless (obsolete) game events are eventually dropped by some *GSS*. An important remark is that our OOS guarantees the game state consistency among all *GSSs*. Put it in other words, the final game state computed by our OOS scheme is not altered while an augmented interactivity is achieved by:

1. dropping obsolete events,
2. permitting different processing orders for non-correlated events at different *GSSs*.

3.2 Evaluation

Our intention now is to report on experimental results obtained from an evaluation of our OOS mechanism. In particular, we measured:

1. the interactivity degree provided by our scheme;
2. the amount of dropped events;
3. the number of rollbacks;
4. the amount of reprocessed events within each single rollback.

To evaluate our OOS scheme, we have simulated a general Mirrored Game Server architecture composed of eight *GSSs*. Without loss of generality, we focused our attention on the event receiving aspect of a given *GSS*, namely GSS_0, pretending that other *GSSs* are sending game events to it.

Table 1 reports the average value and standard deviations of the network latencies that characterize the network links between each sending *GSS* and GSS_0. Based on the literature [2, 9], the transmission delay for each event was obtained on the basis of a lognormal distribution whose parameters were calculated from values reported in Table 1. Also the average event size (200 Bytes), as well as the event generation rate at each *GSS* (modelled through a lognormal distribution with

```
0 procedure OOS-receive-event-procedure() {
1     e := received event;
2     if (e is obsolete)
3        drop(e);
4     else {
5        Event_List := {e_i | e_i already processed ∧
                              e_i correlated to e ∧
                              T_g(e_i) > T_g^p(e)};
6        if (Event_List ≠ NULL) {
7           Rollback(Event_List);
8           Process(e);
9           for (each e_i ∈ Event_List)
10             if (e_i is obsolete)
11                drop(e_i);
12             else Process(e_i);
13          }
14          else Process(e);
15    }
16 }
```

Figure 1: OOS Implementation

Table 1: Configuration of the *GSSs* (ms)

	GSS_1	GSS_2	GSS_3	GSS_4	GSS_5	GSS_6	GSS_7
Latency Avg (ms)	15	40	75	90	80	30	100
Latency Std Dev (ms)	10	15	30	10	20	15	25

an average value of 45 ms and a standard deviation of 10 ms), were inspired by the online gaming literature [2, 9, 24].

Taking inspiration from [2, 9], the *GIT* value was set equal to 150 ms. Further, we conducted our experiments with event traces containing as many as 1000 events for each *GSS*. We considered different event trace configurations, where the probability that a given event is non-correlated to events generated by other players was set equal to, respectively, 50%, 60%, 70%, 80%, 90%. Clearly, the higher the probability of non-correlation, the higher the number of events that will become obsolete during the game evolution. Indeed, a higher non-correlation probability entails that it is more probable the case when an event e^* makes obsolete a preceding one e, as it is more likely that no events have been generated by other players which break the obsolescence relation among e and e^*. Finally, the higher the probability of non-correlation among game events, the lower the number of events that will be subject to rollback.

For a better assessment of the validity of our proposal, in our experiments we contrasted five different synchronization schemes:

1. our OOS scheme;
2. the COS-based Interactivity Restoring scheme (COS-1) described in [10];
3. the COS-based Interactivity Loss Avoidance scheme (COS-2) described in [24];
4. the Time Warp scheme (TW) [17];
5. a traditional Conservative Synchronization (CS) mechanism.

Our intent here is to demonstrate the benefits on the provided interactivity degree attainable by resorting to our OOS approach. To this aim, we first evaluate the five mentioned mechanisms by measuring the percentage of game events arrived at GSS_0 with a *GTD* value larger than *GIT*.

Figure 2 reports the average percentage of *GDTs* above *GIT* as a function of the non-correlation probability. As observable, according to all the experimental configurations, our OOS scheme outperforms the other schemes. It is also worth noticing that in Figure 2 both CS and TW curves lie horizontal and are almost coincident, as standard CS and TW schemes do not take benefits from obsolescence and correlation.

Another proficient tool for evaluating our proposal is represented by the cumulative function of the *GTDs*. This value measures the probability of having a *GTD* lower than a specified value. In substance, the more the line reported in Figure 3 is concentrated in the left side of the chart, the higher is the percentage of events having a *GTD* lower than a certain threshold i.e., the higher the interactivity degree provided by the system. In Figure 3 we report the average value of results obtained from all the considered event trace configurations. As shown in the Figure, OOS outperforms all the other schemes thus guaranteeing an augmented interactivity degree. In essence, these results confirm that:

1. our OOS strategy is far better suited for games w.r.t.

COS schemes, as executing events as soon as they arrive, and then repairing inconsistencies when they are wrong, improve responsiveness;

2. optimistic approaches may take benefits from obsolescence and correlation.

Another interesting measure is concerned with the amount of obsolete events that are dropped to report the interactivity degree within an acceptable value. Indeed, while these dropped events are obsolete and do not influence the final game state, they still are part of the game visual evolution. Thus, to not affect the fluency of the game evolution an excessive dropping of game events should be avoided when possible. Needless to say, only the schemes that are able to drop obsolete events are considered here.

Figure 4 reports the percentage of dropped events using, respectively, COS-1, COS-2 and OOS. As shown in the Figure, OOS reduces the number of discarded events in all the simulated configurations w.r.t. all the other schemes. This effect derives from the fact that our OOS strategy accelerates the event processing, thus diminishing the possibility for an event to become obsolete as fresher events are processed.

We measured also the number of rollbacks needed to maintain the consistency of the game state by comparing the two considered optimistic synchronization algorithms: TW and OOS. This is a fundamental metric to control, as a reduced number of rollbacks allows an augmented interactivity degree. In Figure 5 we show the rollback ratio for TW and OOS. In simple words, we measured the total number of rollbacks in the system divided over the total number of generated events, depending on the non-correlation probability values. It is worth noticing that the TW curve lies horizontal, as standard TW does not take benefits from obsolescence and correlation. It is so obvious that OOS outperforms TW in all the considered scenarios, as OOS does not trigger a rollback procedure for obsolete events.

Finally, Figure 6 reports the average number of re-processed events within a single rollback. Our OOS scheme reduces this value w.r.t. TW, since during a rollback our OOS scheme always avoids the re-execution of those events that become obsolete during the evolution of the game. This clearly diminishes the average number of game events that need reprocessing within a single rollback.

4. RELATED WORK: A COMPARISON

Several synchronization mechanisms have been recently presented in the online gaming literature devised to guarantee a uniform view of the game state, when it is replicated across different game nodes distributed throughout the Internet [5, 6, 7, 19, 21, 22]. Among these, similarly to our OOS scheme, some of them take inspiration from the optimistic Time Warp algorithm that adopt a detect and correct strategy [5, 6, 21, 22].

For instance, the *trailing state synchronization* algorithm, presented in [5, 6] and devised for the support of online games, is based on the idea of locally maintaining at all *GSSs* a fixed number of copies of the game state, each of which is kept at a different simulation time. In essence, each copy of the game state is associated to a particular execution; each execution is delayed for a fixed time interval. Thus, inconsistencies are identified by comparing the leading state (where game events are processed optimistically without any delay) with the game states obtained by delaying execution of a fixed value equal to Δt. If an inconsistency is discovered, a rollback is performed by resorting to the game state of the delayed execution. This ensures that all game events are processed in the proper order.

In [21, 22], instead, an approach has been devised which uses the *local lag* control mechanism combined with a modified Time Warp (executed only when necessary). Taking in mind the important trade-off existing between responsiveness and consistency, the authors propose to deliberately decrease the responsiveness of the application in order to eliminate short-term inconsistencies. In essence, each received game event is delayed for a certain amount of time (local lag) so as to reorder the received events and minimize inconsistencies. However, if a game event arrives later than the temporal delay identified by the local lag, then the approach resorts to rollback-based recovery schemes to correct the computation.

The main difference between other rollback-based schemes and our OOS mechanism is that we are able to keep under control the number of events to be rolled back thanks to the use of the notion of obsolescence and correlation.

Alternatively, other approaches exist which simply assume that delaying the game event processing activity for a fixed amount of time may be sufficient to guarantee a uniform evolution of the game state at different nodes of the architecture without any need to resort to rollback.

Following this idea, a synchronization approach has been presented in [7, 19] which has been exploited for the deployment of MiMaze. This mechanism is an optimistic version of the well known conservative bucket synchronization algorithm. The approach consists in assuming that a processing deadline (time bucket) exists before which all game events have to be received for a correct evolution of the game. In essence, the idea behind this scheme is to process game events at the end of the time bucket. Simply put, at the end of the bucket game events are ordered and then processed. If some game events are not received before the time bucket expiration, dead reckoning techniques are exploited to compensate events losses.

A main drawback with a time bucket approach is that dead reckoning may not ensure the consistency of the distributed game state. Thus, game inconsistencies may arise if the bucket size is set equal to a small value. On the other hand, using a large value of the bucket size may induce a severe degradation of the system performances. Further, a more complex problem is that of fitting the bucket size with an unstable condition of the Internet where large values of jitter delays may be experienced. On the contrary, we react to network congestions, packet losses, transmission delays and delay jitters by employing a discarding mechanism for obsolete events and by resorting to a delivery strategy based on the notion of correlation.

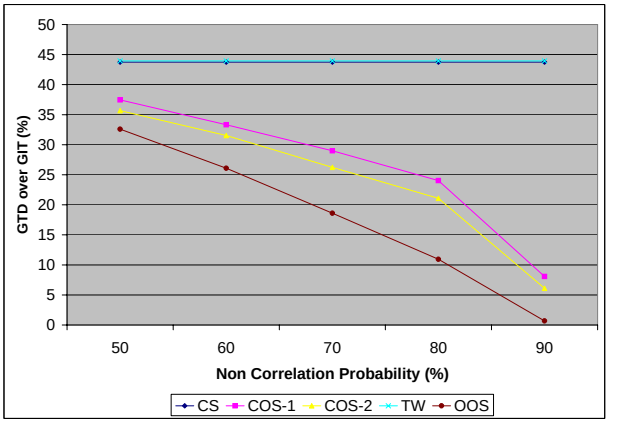

Figure 2: Percentage of Events with *GTD* over *GIT*

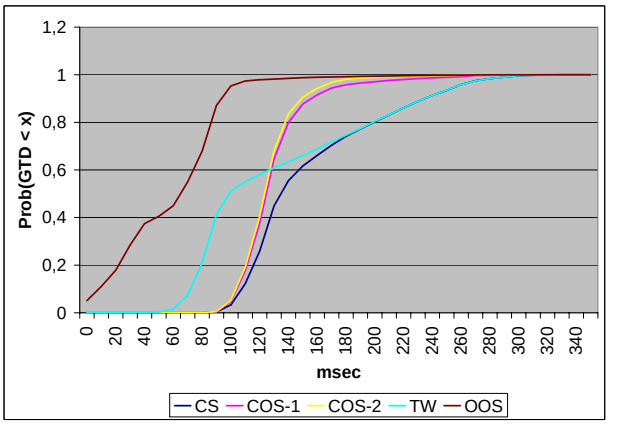

Figure 3: Cumulative Function of the *GTDs*

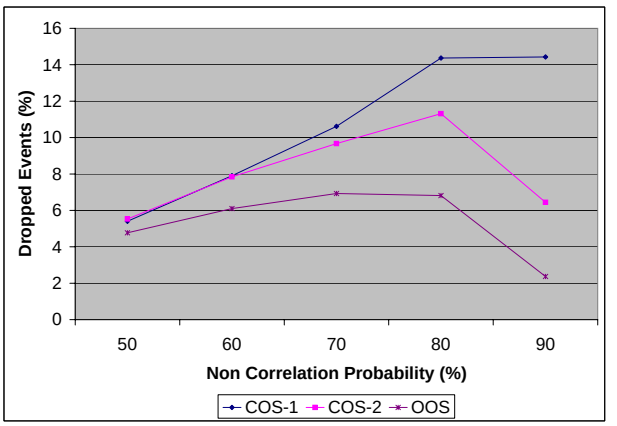

Figure 4: Percentage of Discarded Events

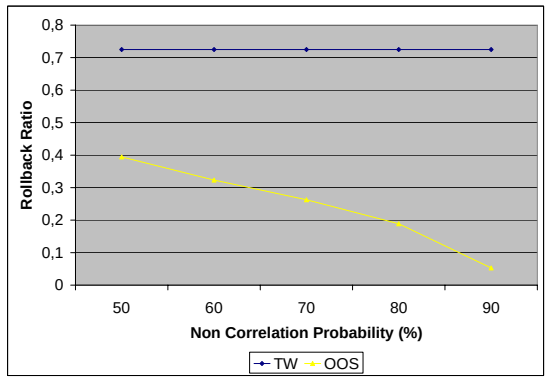

Figure 5: Rollback Ratio

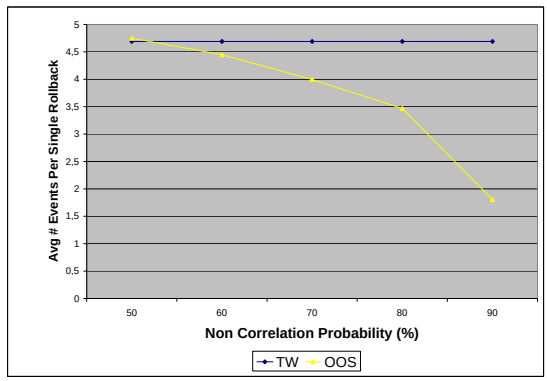

Figure 6: Number of Re-Processed Events per Rollback

5. CONCLUSIONS

The growth in the popularity of highly interactive MMOGs has increased the importance of a better understanding of new protocols and schemes to support a vast amount of players connected to an online game. In this sense, a key issue is represented by the capability of rapidly synchronizing distributed mirrored game servers that manage a redundant copy of the game state.

Our work has confirmed that an optimistic synchronization approach, coupled with the notions of obsolescence and correlation, is well suited for MMOGs, as it allows game servers to better follow the evolution of fast paced games.

To conclude, it is worth mentioning that our mechanism has been recently embedded into a MMOG we developed to allow customers to play a car racing game over the Web [3]. Preliminary results have confirmed that resorting to our OOS scheme improves the interactivity degree with respect to the use of other synchronization mechanisms which were previously employed to support the game (e.g., COS-1).

6. ACKNOWLEDGMENTS

We wish to thank the Italian M.I.U.R. (Interlink) for the partial financial support to our research. Many thanks also to the anonymous referees of the ACE Conference for their helpful comments on an earlier version of this paper.

7. REFERENCES

[1] I. S. 1278.2-1995. IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles, 1995.

[2] M. Borella. Source models for network game traffic. *Computer Communications*, 23(4):403–410, February 2000.

[3] S. Cacciaguerra, S. Ferretti, M. Roccetti, and M. Roffilli. Car racing through the streets of the web:

a high-speed 3d game over a fast synchronization service. In *Proceedings of International ACM World Wide Web 2005 Conference, Poster Track*, Chiba, Japan, May 2005.

[4] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[5] E. Cronin, B. Filstrup, S. Jamin, and A. Kurc. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, 23(1):7–30, May 2004.

[6] E. Cronin, B. Filstrup, A. Kurc, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of the 1st Workshop on Network and System Support for Games*, pages 67–73. ACM Press, 2002.

[7] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Network Magazine*, 13(4), July/August 1999.

[8] R. Drummong and O. Babaoglu. Low-cost clock synchronization. *Distributed Computing*, 6(3):193–203, 1993.

[9] J. Farber. Network game traffic modelling. In *Proceedings of Netgames'02*, pages 53–57. ACM Press, 2002.

[10] S. Ferretti and M. Roccetti. The design and performance of a receiver-initiated event delivery synchronization service for interactive multiplayer games. In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (Game-On 2003)*, London, UK, November 2003.

[11] S. Ferretti and M. Roccetti. On designing an event delivery service for multiplayer networked games: An approach based on obsolescence. In *Proceedings of IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2003)*, Honolulu, HI, August 2003.

[12] S. Ferretti and M. Roccetti. Event synchronization for interactive cyberdrama generation on the web: A distributed approach. In *Proceedings of 13th International World Wide Web Conference (WWW 2004)*, volume WWW2004 Poster Track, New York, NY, May 2004.

[13] S. Ferretti and M. Roccetti. A novel obsolescence-based approach to event delivery synchronization in multiplayer games. *International Journal of Intelligent Games and Simulation*, 3(1):7–19, March/April 2004.

[14] S. Ferretti, M. Roccetti, and S. Cacciaguerra. On distributing interactive storytelling: Issues of event synchronization and a solution. In *Proceedings of the 2nd International Conference on Technologies for Digital Storytelling and Entertainment (TIDSE 2004), LNCS 3105*, pages 219–231, Darmstadt, Germany, June 2004.

[15] R. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., 1999.

[16] R. Gusella and S. Zatti. The accuracy of clock synchronization achieved by tempo in berkeley unix 4.3bsd. *IEEE Transactions of Software Engineering*, 15(7):47–53, July 1989.

[17] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[18] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, pages 96–107. IEEE, March 2004.

[19] K. Lee, B. Ko, and S. Calo. Adaptive server selection for large scale interactive online games. In *Proceedings of the 14th international Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 152–157. ACM Press, 2004.

[20] F. Li, L. Li, and R. Lau. Supporting continuous consistency in multiplayer online games. In *Proceedings of the 12th annual ACM international conference on Multimedia (MULTIMEDIA '04)*, pages 388–391. ACM Press, 2004.

[21] M. Mauve, S. Fischer, and J. Widmer. A generic proxy system for networked computer games. In *Proceedings of Netgames'02*, pages 25–28. ACM Press, 2002.

[22] 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, February 2004.

[23] D. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[24] C. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. On maintaining interactivity in event delivery synchronization for mirrored game architectures. In *Proceedings of the 1st IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'04)*, pages 157–165, Dallas, USA, November 2004.

[25] C. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. A rio-like technique for interactivity loss avoidance in fast-paced multiplayer online games: a preliminary study. *ACM Journal of Computer in Entertainment*, 3(2), April-June 2005.

[26] L. Pantel and L. Wolf. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st Workshop on Network and System Support for Games*, pages 79–84. ACM Press, 2002.