Fast Synchronization of Mirrored Game Servers: Outcomes from a Testbed Evaluation

Stefano Ferretti, Marco Roccetti, Alessio la Penna
Department of Computer Science
University of Bologna
Mura Anteo Zamboni, 7
40127 Bologna - Italy
E-mail: {sferrett, roccetti, lapenna}@cs.unibo.it

KEYWORDS
MOGs, synchronization, consistency, responsiveness, interactivity.

ABSTRACT

The deployment of online games over the Internet encompasses the use of novel, smart strategies able to guarantee, on one side, a high level of responsiveness in the game system and, one the other side, the consistency of the computed game state. The use of mirrored game servers has been recognized as a scalable, fault tolerant architectural solution for the support of Multiplayer Online Games (MOGs). We developed a new optimistic synchronization scheme devised for MOGs. The intelligence behind our scheme is based on the idea of exploiting two new notions of obsolescence and correlation among game events. These notions allow one to speed up the synchronization among replicated servers, while maintaining the consistency of the game state. In this work, we report on an experimental assessment based on a real implementation of a mirrored game server architecture, deployed over the Internet, that exploits our synchronization algorithm. Specific attention is devoted to the experimental assessment. Results show the viability and the efficacy of our approach.

INTRODUCTION

Providing support to Multiplayer Online Games (MOGs) is one of the most striking issues in the field of distributed multimedia systems. According to these games, several aspects arise that must be tackled from different angles. Employed software architectures must be scalable and tolerant to faults. Algorithms utilized to deliver and manage game data must be fast, reliable, fair and cheat-proof. Protocols that enable game nodes to interact should be general and effective.

Prompted by these hard requirements, researchers and industries are working aimed at finding new solutions for supporting MOGs over best effort, wide area networks. Noteworthy advancements have been done in this direction [Borella 2000, Cronin 2002, Ferretti et al 2006, Knutson 2004, Mauve 2004, Mueller et al. 2005, Palazzi et al 2006]. Yet, the problem is far from being solved.

From an architectural point of view, a new distributed solution has been identified as a viable approach to support MOGs. This architectural solution is a hybrid between classic client/server approaches (which typically lack fault-tolerance and scalability) and peer-to-peer approaches (which represent promising solutions but may impose that high numbers of messages are sent through the network when multicast-based delivery strategies cannot be employed). These hybrid approaches are commonly referred as mirrored game servers architectures [Cronin 2002, Mauve 2004, Palazzi et al. 2006]. In essence, several Game State Servers (GSSs) are distributed over the Internet and maintain a replicated game state. Clients connect to one GSS and communicate only with it. Moreover, GSSs synchronize themselves to maintain a consistent game state.

Needless to say, in this context, the synchronization algorithm employed among GSSs plays a fundamental role for the overall performances of the game system. Indeed, this algorithm must guarantee the consistency of the replicated game state while enabling the distribution of “fresh” game data in a very quick way. As a matter of facts, interactivity (thought as the degree of responsiveness provided by the system) is the main issue in online games, especially in fast-paced games. However, the need for a reliable and totally ordered delivery of game events in synchronization algorithms could affect the level of provided interactivity.

With this in view, we have recently proposed a synchronization scheme which is based on the idea of exploiting the semantics of game events in order to relax (when possible) reliability and ordering requirements. This approach is accomplished in order to provide an augmented interactivity degree while maintaining game state consistency [Ferretti & Roccetti 2005, Ferretti et al. 2006]. Specifically, the intelligence behind our algorithm consists in exploiting two new notions of obsolescence and correlation among game events. These notions provide GSSs with the ability of classifying and characterizing messages coming from players. As a result, GSSs are enabled to discard superseded data and to process game events according to different orders, at different hosts, when this does not introduce inconsistencies in the distributed computation. The synchronization scheme is optimistic, i.e., events are processed as soon as they are received and inconsistencies in the distributed computation are corrected by rolling back erroneous computations.

The considered scheme was verified through extensive simulations. Results showed the viability of our approach [Ferretti & Roccetti 2005, Ferretti et al. 2006]. Obviously, simulation results are particularly meaningful, since they are repeatable and simulation settings can be varied so as to assess the scheme according to different scenarios. Nevertheless, there are several assumptions, which are typically made during simulations, that may be flawed in a real context. For instance, the assumption that nodes' physical clocks are perfectly synchronized is quite obvious in a simulated scenario. In the real world, perfect physical clocks synchronization is hardly met. Thus, a question arises whether such an issue may affect the system performances in some way. Other examples are concerned with the clock drift rate of nodes and differences in the computational capabilities. Summing up, a main aspect of interest in our research was that of assessing our optimistic synchronization scheme in a real functioning mirrored game server architecture.

With this in view, we have implemented a real distributed mirrored game server architecture composed of three GSSs deployed in the Internet. Specifically, two hosts were placed in Italy, while the third one was placed in California, U.S. Each GSS was equipped with our optimistic synchronization algorithm to maintain a vision of the game state which is consistent with that maintained by other GSSs. In this paper, we report on technical issues related to the real deployment of the game system and show results obtained through this real experimental assessment.

The main outcomes of our study can be summarized as follows. First, the assessment demonstrated that our approach could really improve the responsiveness degree provided by the system, while guaranteeing a uniform view of the game state evolution at different replicated servers. Second, having synchronized physical clocks and regulating the clock drift rate are two important aspects to face with in MOGs. Indeed, we noticed highly drift rates among clock nodes. To solve these issues, we exploited a periodic physical clock synchronization approach so as to correct effects due to the drift rate. Third and final, we found out that the critical node, placed at higher distance from the other two GSSs, characterizes the overall performances of the game system, as claimed in [Brun et al. 2006]. This result has an obvious explanation. Even if the synchronization scheme is optimistic, late events received by the critical node force other nodes to rollback their computations very often.

The remainder of this paper is organized as follows. Next Section reports on some background which is useful to introduce our optimistic synchronization algorithm. Then, in the third Section, we report on issues concerned with the problem of developing and deploying a real mirrored game server architecture over the Internet. We also present in detail the experimental testbed we exploited to assess our approach. Obtained results are then showed and discussed in the fourth Section. Finally, some conclusions are provided in the last Section.

BACKGROUND

In this Section, we report on main results obtained in the literature which are useful for the presentation of our work. In particular, first, we discuss the need for scalable and fault tolerant distributed solutions for the support of MOGs. Second, we put emphasis on the existing trade off between the problem of guaranteeing a consistent game evolution at all nodes in the game system and that of guaranteeing a responsive evolution of the game. Third and final, we present an optimistic synchronization algorithm that may successfully trade between these two issues.

Mirrored Game Server Architectures

Several works have been accomplished with the aim of finding the most suitable architectural solutions for the support of MOGs [Cronin 2004, Ferretti 2005, Ferretti and Roccetti 2005, Ferretti & al. 2006, Mauve 2004, Palazzi 2006, Wright 2004]. Among classic completely centralized (i.e., client/server) and fully decentralized (i.e., peer-to-peer) solutions, an interesting proposal has been recognized which is based on the idea of utilizing several *mirrored* GSSs which are geographically dispersed over the net. Each player connects to its *nearest* GSS and communicates with it in a classic client/server style. In turn, each GSS is interconnected with all other GSSs resembling a P2P architecture [Diot 1999, Knutsson 2004]. Each GSS maintains a local view of the game state. Thus, with each new action performed by each player, its corresponding GSS collects the generated event, notifies it to other GSSs, updates the game state locally and, finally, communicates the newly computed game state to their connected players.

It has been demonstrated that the use of a mirrored game server architecture results as a profitable solution for supporting highly distributed and crowded MOGs. Indeed, only subsets of users connect to the same GSS. This way, network and computational overheads at the server-side are reduced. Moreover, the system is particularly tolerant to faults, since GSSs are not single point of failures in the system. Finally, this scheme lets a huge number of players to join the same game, since different users may be connected to different GSSs while participating to the same game session.

Consistency vs Interactivity: The Need for an Intelligent Synchronization Scheme

The use of multiple GSSs, each of which manages a replicated state of the game, forces GSSs to synchronize themselves in order to maintain a local, consistent vision of the game evolution. Several schemes exist for consistency maintenance. The problem with most of these mechanisms is that they may require the use of some sort of synchronization barriers [Palazzi 2006]. While useful in several contexts of distributed systems, most of the classic synchronization schemes fail when employed on MOGs. Basically, synchronization approaches aim at providing ways to totally order game events generated during the game evolution. Hence, a main issue is that of identifying a proper ordering scheme for game events. Several solutions exist, ranging from simple turn-based schemes to timestamp-based orders [Ferretti 2005, Fujimoto 1999].

According to the timestamp order, events are ordered and processed based on an order obtained by exploiting timestamps inserted within messages. As a matter of facts, it results that a game event ordering scheme, based on timestamps associated to game events produced by players, is a very suitable one to ensure that the game advances in (scaled) real-time [Fujimoto 1999]. The use of a timestamp order ensures that all GSSs process game events in the same way, thus guaranteeing that GSSs pass through same game state updates. 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 [Cronin 2004, Diot 1999, Fujimoto 1999, Lee 2004, Li 2002, Mauve 2004].

In substance, a tradeoff relationship exists between full consistency and interactivity maintenance. Efficient, intelligent and fast synchronization schemes are needed so as to let GSSs to effectively collect game events produced by players, compute correct game states and notify players with fresh gaming information in a responsive way. More specifically, for each new game event, an interval of time can be measured between its generation and its arrival at a given node. We term this resulting value Game Time Difference (GTD). To have interactivity, new, consistent game updates should be delivered to all participants without surpassing a significant Game Interactivity Threshold (GIT). The main aim of a MOG synchronization algorithm should be that of keeping GTDs of game events lower than GIT [Ferretti 2005, Palazzi et al. 2006].

A Fast Optimistic Obsolescence-based Synchronization Algorithm

With both the needs for interactivity and consistency in view, we have recently proposed a new optimistic synchronization algorithm which is based on the idea of exploiting the semantics of game events produced during the game [Ferretti & Roccetti 2005, Ferretti & al. 2006]. In particular, the mentioned approach is based on the notions of correlation and obsolescence among game events. GSSs are supplied with a smart and fast method for classifying and characterizing messages coming from players. In the following, we provide an informal explanation of these two notions.

Two game events are said to be *correlated* if different orders of execution of these game events lead to different game states. Examples of correlated game events are those that act on same game elements in a game world. Instead, independent movements of different virtual characters are examples of non-correlated game events.

The notion of *obsolescence* is a very intuitive one. Simply put, new game events may render old information as obsolete. For instance, the position of a given virtual character at a given time may become an obsolete information when the character moves to another position. In general, many situations exist according to which fresher game events annul the importance of previous events. Indeed, cases exist when processing a new game event (say e_{new}) without considering the first one (say e_{old}) leads to the same final state that would be reached if both events were processed in the correct order (i.e., e_{old} becomes obsolete). As a matter of facts, while simple to understand, obsolescence is not a naïve concept. In fact, cases exist when obsolescence relation cannot be applied among two events e_{old} and e_{new}. In particular, a given event e_{old} cannot be considered as made obsolete by e_{new}, when other events e_i correlated to e_{old} have been generated within the time interval comprised between the generation of e_{old} and e_{new}. These game events e_i may alter the evolution of the plot thus making unapplicable the notion of obsolescence. Deeper details about obsolescence and correlation may be found in [Ferretti 2005, Palazzi et al. 2006].

Based on these concepts of correlation and obsolescence, we have devised and developed a novel *Optimistic Obsolescence based Synchronization* (OOS) algorithm [Ferretti & Roccetti 2005, Ferretti et al. 2006]. The idea behind the approach is that ordering and reliability requirements can be relaxed for, respectively, non-correlated and obsolete game events. Indeed, obsolete game events can be discarded during the notification and processing activities, thus fastening the event delivery among GSSs. Moreover, to provide players with a uniform evolution of the game, it is enough that only 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.

Based on obsolescence and correlation, OOS is an optimistic approach which is based on the well-known Time Warp algorithm [Jefferson 1985]. Specifically, according to OOS, aften reception of a new game event e, each GSS verifies if e may be identified as obsolete. In this case, e is dropped. Otherwise, a check is carried out to control whether any other game events e_i, correlated to e and generated after e, have been already processed at that GSS. If this check succeeds, then a rollback procedure is performed where all these events e_i are rolled back. At this point, e is processed, followed by the execution of all those rolled back events which are not obsolete. In fact, obsolete events are discarded during the rollback. Only when e is not obsolete and no events correlated to e have been processed out of order, e is directly processed. The interested reader may refer to [Ferretti and Roccetti 2005, Ferretti & al. 2006] for a complete discussion of the OOS scheme.

DEVELOPING A REAL DISTRIBUTED MIRRORED GAME SERVER ARCHITECTURE

What Happens in a Real, Wide Area Net?

The synchronization algorithm OOS described in the previous Section was widely assessed through the use of simulations [Ferretti and Roccetti 2005, Ferretti & al. 2006]. Of course, simulations have the great benefit of outputting consistent and verifiable data. However, while obtained results were particularly meaningful and sufficiently adequate to confirm the goodness of our approach, a main open issue was that of assessing our synchronization algorithm in a real, fully operational mirrored game server architecture.

Indeed, with the aim of focusing on technicalities of the specific problem being considered, often in simulations researchers make proper assumptions on their simulated system. As a matter of facts, in the real world often these assumptions fail or become very difficult to address. Specifically, in our considered scenario, assumptions which are definitively not valid are the following ones: having i) a perfect synchronization among nodes' physical clocks, ii) the same, or a negligible, clock drift rate, iii) similar computational capabilities of nodes, iv) the same workload for each node in the system.

With this in view, we have developed, deployed on the Internet and extensively tested a real functioning mirrored game server architecture.

Some Implementation Details

Our implementation of OOS in real GSSs is based on a receiver initiated event delivery scheme, built over UDP. Basically, according to this approach negative acknowledgements (NACKs) are exploited to provide reliability in the event notification among GSSs, only if the considered game events are not become obsolete [Ferretti 2005]. This way, reliability constraints are relaxed for those game events that result obsolete. This approach has the prominent advantage of reducing messages sent throughout the network and fasten the event delivery activity.

As already mentioned, our synchronization algorithm exploits an order based on timestamps associated to game events to guarantee game state consistency. Therefore, during the development of a real mirrored game server architecture, synchronization of physical clocks of nodes becomes an important issue to address. Not only, also the clock drift rate must be kept as a negligible value to have that the pace of the game advancements are similar to all nodes in the system.

In order to regulate physical clock synchronization and annul dangerous effects due to possibly high clock drift rates, we have implemented a low level synchronization scheme, based on that proposed by Cristian [Cristian 1989], that continuously runs during the experimentations (i.e., the game evolution), able to dynamically correct clocks' skew of GSSs.

Experimental Testbed

To evaluate our scheme, we have deployed over the Internet a real mirrored game server architecture comprising three different GSSs. As shown in Figure 1, a first GSS was placed at the Department of Computer Science, University of California Los Angeles. Other two GSSs were placed, respectively, at the Department of Computer Science of the University of Bologna, Italy, and at the detached study course of Computer Science in Cesena, which is a structure of the Multi-Campus Project of the University of Bologna. Average latencies among nodes constituting this mirrored developed architecture are shown in Figure 1.

It is important to notice that even if in our testbed two GSSs were placed at a really short network distance, it is the critical node (i.e., the one with higher latency with respect to other ones) that mainly affects the performances of the system [Brun 2006]. Indeed, even with a fast synchronization scheme, game events generated by the critical node will slow down the computation of new consistent and correct game states.

As already mentioned, due to our goal of assessing our synchronization algorithm in a real mirrored game architecture, fully operational GSSs were installed on the network. However, clients connected to GSSs were emulated. This enabled us to monitor GSSs' behavior under very different gaming conditions. The event generation rate at each emulated client was set to follow a lognormal distribution, as inspired by literature on games, and to vary from a normal traffic situation to an intensively loaded one [Borella 2000, Farber 2002]. In particular, different experiments have been conducted with an Average Departure Time (ADT, i.e., the average time interval that passes between the generation of two subsequent events) at each emulated client varying from 30 msec (intense game traffic) to 45 msec (moderate game traffic), while having the standard deviation constantly equal to 10 msec. We emulated 10 different clients connected to each GSS. Finally, the average game event size was set equal to 200 Bytes.

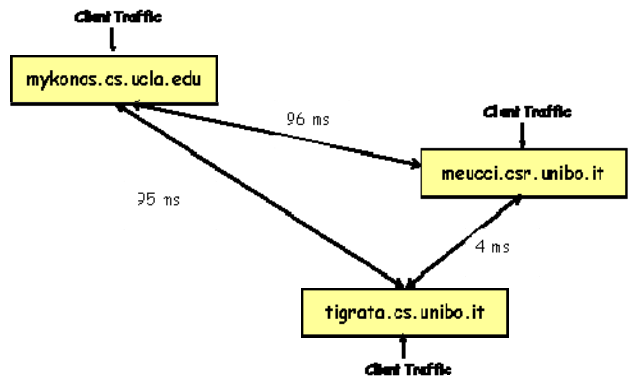

Figure 1: System Model

To assess the efficacy of our approach in intense gaming scenarios, we set the GIT value equal to 200 msec. This setting for the interactivity threshold corresponds to a configuration related to responsiveness requirements of fast-paced games.

The probability that a given event is non-correlated to other events was set to vary from 50% to 90%. It is worth noticing that this metric can be equivalently thought as a metric that regulates the probability that an event makes obsolete preceding ones. Indeed, a higher non-correlation probability entails a higher probability that a new event renders obsolete previous ones generated by the same players, since a lower amount of game events will interfere with this relation among the two events. Another consideration worth of mention is that a high probability of having obsolete events represents a realistic scenario for a vast plethora of possible games (i.e., adventure, strategic, car race, flight simulator, etc.). For instance, it can be argued that most of the events generated in games are just independent moves. In other words, critical (correlated) game events that cannot become obsolete have to be considered only sporadically, such as during collisions or shots, and typically represent even less than the 10% of the whole set of game events. An extensive discussion of this claim has been reported in [Palazzi et al. 2006].

EXPERIMENTAL RESULTS

Measures of Interest

In this study we were interested in evaluating the following measurements:

- The average number of game events with a GTD higher than GIT. Obviously, the lower this average value, the higher the level of interactivity and responsiveness provided by the system.
- The amount of activated rollback procedures. Also in this case, the lower this value the lower the computational overhead added in the synchronization process and the higher the responsiveness degree.
- The amount of obsolete events which are dropped by our scheme.

To measure the benefits introduced by the use of the notions of obsolescence and correlation, we contrasted our OOS scheme with an implementation of the well known Time Warp [Jefferson 1985].

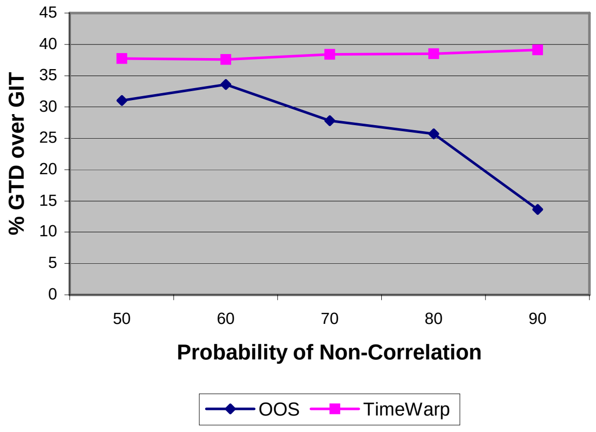

Figure 2: Average Percentage of Events with GTD over GIT, ADT = 30 msec

Outcomes

Figure 2 and 3 report the average amount of events that experienced a GTD value that is higher than GIT. The two charts correspond to different ADT values, i.e., different paces of generation of game events at clients. As can be noticed, our OOS scheme performs better than Time Warp, thus ensuring a higher interactivity degree. Specifically, with our scheme, when the ADT is set equal to 45% and the probability of non-correlation is above 70%, only a negligible number of game events experience a GTD which is above the interactivity threshold.

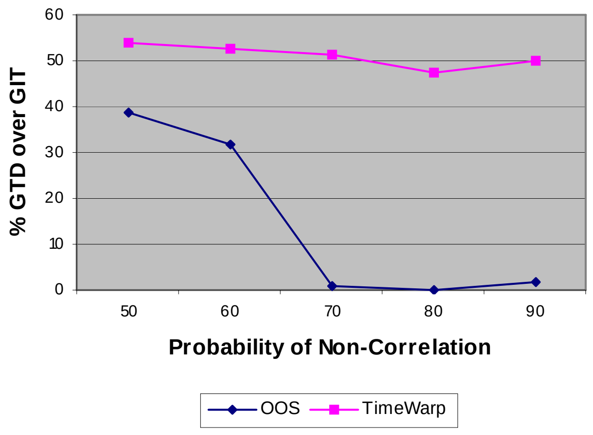

Figure 3: Average Percentage of Events with GTD over GIT, ADT = 45 msec

Figures 4 and 5, instead, report the average rollback ratio experienced by the two considered optimistic synchronization algorithms during our tests. In substance, each chart reports an averaged number obtained by dividing the total number of performed rollbacks over the total number of generated events. As expected, since obsolete events are dropped during the event notification activity, the need for rollbacks diminishes when resorting to our OOS scheme.

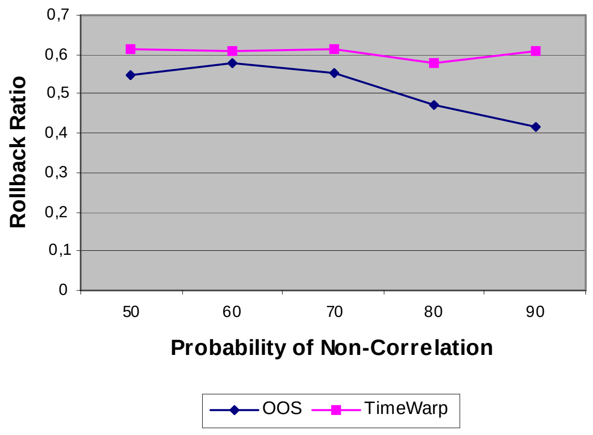

Figure 4: Rollback Ratio, ADT = 30 msec

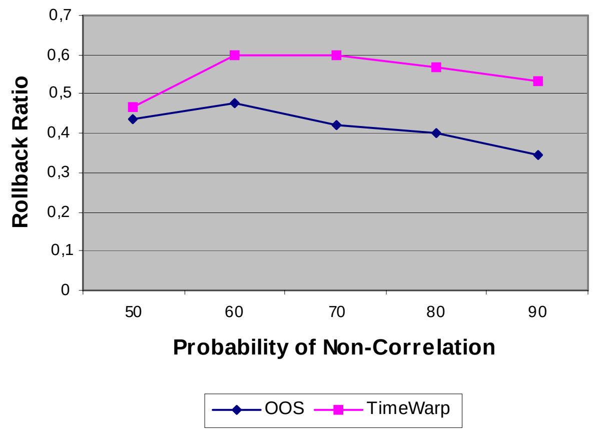

Figure 5: Rollback Ratio, ADT = 45 msec

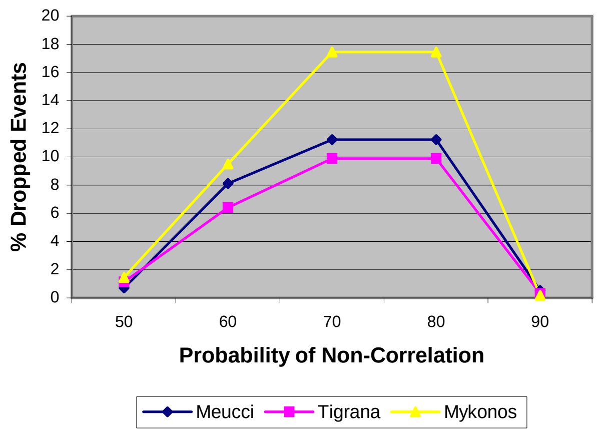

Figure 6: Percentage of Dropped Events, ADT = 30 msec, Meucci refers to the host in Cesena, Tigrana refers to the host in Bologna, Mykonos refers to the host in Los Angeles

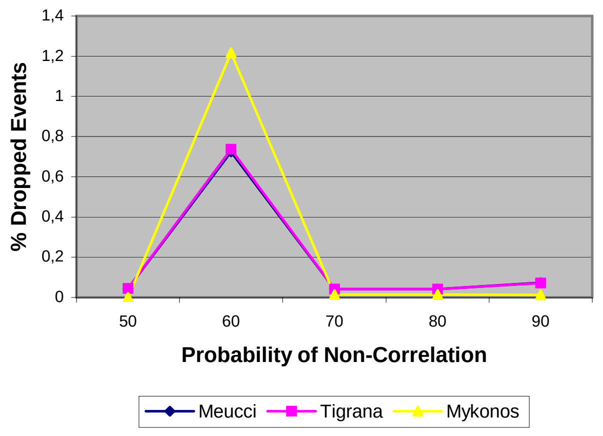

Figure 7: Percentage of Dropped Events, ADT = 45 msec, Meucci refers to the host in Cesena, Tigrana refers to the host in Bologna, Mykonos refers to the host in Los Angeles

It can be noticed that when the probability of obsolescence grows up to 90%, the rollback ratio does not decrease linearly. Probably, this effect is due to the fact that events locally generated at a given GSS are always timestamped, notified to other GSSs, and then optimistically processed at that GSS as soon as they are received from the client. In other words, events generated by players connected to that GSS are, on average, processed before those (with similar timestamps) generated by clients connected to other GSSs. Hence, a given ratio of rollbacks is inevitable when resorting to optimistic synchronization schemes. Nevertheless, our scheme improves this considered performance metric.

Figures 6 and 7 show the average percentage of obsolete events, dropped at each GSS during our tests, depending on the considered ADT. As a first obvious consideration, it results that the U.S. node (i.e., Mykonos, see also Figure 1) drops a higher amount of game events, since there are higher network latencies to reach that node with respect to the other ones placed in Italy (i.e., Meucci and Tigrana, see also Figure 1). It is worth noticing that in both charts, the number of dropped events initially grows with the percentage of non correlation. This can be explained with the fact that, the more the number of available events that become obsolete (i.e., non correlated to other ones generated by other players), the more each GSS is able to drop obsolete events for an augmented interactivity. Surprisingly, once surpassed a given percentage of non correlation probability, the number of dropped events decreases considerably. This is probably due to the fact that, with a high non correlation (i.e., higher obsolescence) probability, our scheme is able to provide a high level of interactivity; thus, game events are typically processed before they can become obsolete.

Obviously, performances improve with a higher ADT, since a slower pace of generation of game events imply a smaller number of game events to be transmitted, managed and processed per unit of time.

As a final consideration, these results confirm those obtained thought the use of simulations. Thus, we are able to conclude that our scheme can be of real service to guarantee interactive gaming experiences in massively distributed MOGs while maintaining the coherence of the view of the game evolution at different, mirrored game servers.

CONCLUSIONS

In this paper, we have reported on a real experimental campaign we have built so as to assess our OOS algorithm.

As a main outcome of our evaluation, we found out that OOS is of real service when MOGs are deployed over scalable mirrored game server architectures. Indeed, OOS could really improve the responsiveness degree provided by the system, while maintaining game state consistency.

We also observed that physical clocks' nodes may really drift apart during the game evolution. To face this issue, a periodic physical clock synchronization approach should be utilized to maintain short clock skews among nodes.

ACKNOWLEDGMENTS

We wish to thank the Department of Computer Science of UCLA for their technical support during the experimental assessment. This work is financially supported by the Italian M.I.U.R. under the MOMA and DAMASCO initiatives.

REFERENCES

M.S. Borella. 2000. “Source Models for Network Game Traffic”, *Computer Communications*, vol. 23, no. 4, 2000, pp.403-410.

J. Brun, F. Safei, P. Boustead. 2006. “Server Topology Considerations in Online Games”, in *Proceedings of Netgames’06*, Singapore, October 2006.

F. Cristian. 1989. “Probabilistic Clock Synchronization”, Distributed Computing, Vol. 3, No. 3, 1989, 146-158.

E. Cronin, A. R. Kurc, B. Filstrup, S. Jamin. 2004. “An Efficient Synchronization Mechanism for Mirrored Game Architectures”, *Multimedia Tools and Applications*, vol.23, no.1, 2004, pp.7-30.

J. Farber. 2002. “Network Game Traffic Modelling”, in *Proceedings of NetGames2002*, Braunschweig, Germany, 2002, pp.53-57.

S. Ferretti & M. Roccetti. 2005. “Fast Delivery of Game Events with an Optimistic Synchronization Mechanism in Massive Multiplayer Online Games”, in *Proceedings of ACM SIGCHI International Conference on Advances in Computer Entertainment Technology* (*ACE 2005*), Valencia (Spain), ACM, June 2005, 405-412.

S. Ferretti, M. Roccetti & C.E. Palazzi. 2006. “An Optimistic Obsolescence-Based Approach To Event Synchronization For Massive Multiplayer Online Games”, *International Journal of Computers and Applications*, ACTA Press, February 2006, to appear.

R. Fujimoto. 1999. “Parallel and Distribution Simulation Systems”, John Wiley Inc.

D.R. Jefferson, “Virtual Time”, *ACM Transaction on Programming Languages and Systems*, vol. 7, no. 3, 1985, pp. 404-425.

B. Knutsson, L. Honghui, X. Wei, B. Hopkins. 2004. “Peer-to-peer support for massively multiplayer games”, in *Proceedings of the Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* (*INFOCOM 2004*), March 2004, 96-107.

K. Lee, B. Ko, S. Calo. 2004. “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 (NOSSDAV 2004)*, ACM Press, 2004, pp. 152–157.

F. Li, L. Li, and R. Lau. 2004. “Supporting continuous consistency in multiplayer online games”, in *Proceedings of the 12th annual ACM international conference on Multimedia (MULTIMEDIA ’04)*, ACM Press, 2004, pp. 388–391.

M. Mauve, J. Vogel, V. Hilt, W. Effelsberg, “Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications”, *IEEE Transactions on Multimedia*, vol.6, no.1, 2004, pp. 47-57.

J. Muller, S. Gorlatch. 2005. “Rokkatan: scaling an RTS game design to the massively multiplayer realm”, *ACM Computers in Entertainment*, ACM Press, Vol. 4, Is. 3, July 2006.

C.E. Palazzi, S. Ferretti, S. Cacciaguerra & M. Roccetti. 2006. Interactivity-Loss Avoidance in Event Delivery Synchronization for Mirrored Game Architectures, in *IEEE Transactions on Multimedia*, IEEE Signal Processing Society, Vol. 8, No. 4, August 2006, 874-879.

S. Wright, S. Tischer, “Architectural Considerations in Online Game Services over DSL Networks”, in *Proceedings of IEEE International Conference on Communications - (ICC'04)*, IEEE Communications Society, Paris, France, 2004, pp.1380-1385.