Consistency in Replicated Continuous Interactive Media

Martin Mauve
Praktische Informatik IV
University of Mannheim
L15, 16
68131 Mannheim
mauve@informatik.uni-mannheim.de

ABSTRACT

In this paper we investigate how consistency can be ensured for replicated continuous interactive media, i.e., replicated media which change their state in reaction to user initiated operations as well as because of the passing of time. Typical examples for this media class are networked computer games and distributed VR applications. Existing approaches to reach consistency for replicated discrete interactive media are briefly outlined and it is shown that these fail in the continuous domain. In order to allow a thorough discussion of the problem, a formal definition of the term consistency in the continuous domain is given. Based on this definition we show that an important tradeoff relationship exists between the responsiveness of the medium and the appearance of short-term inconsistencies. Until now this tradeoff was not taken into consideration for consistency in the continuous domain, thereby severely limiting the consistency related fidelity for a large number of applications. We show that for those applications the fidelity can be significantly raised by voluntarily decreasing the responsiveness of the medium. This concept is called local lag. It enables the distribution of continuous interactive media that are more vulnerable to short-term inconsistencies than, e.g., battlefield simulations. We prove that the concept of local lag is valid by describing how local lag was successfully used to ensure consistency in a 3D telecooperation application.

Keywords

Consistency, Replicated Continuous Interactive Media

INTRODUCTION

During the past decade the problem of consistency in synchronous CSCW applications (e.g., shared text editors) has been the focus of attention for many research groups. Most work in this area has been done on the consistency of *replicated discrete interactive media* [15,3,14], i.e., replicated media which change their state only in response to user initiated operations. In the last three to four years, however, a broad variety of applications has evolved which employ *replicated continuous interactive media*, e.g., networked computer-games, multi-user virtual reality, distributed simulations for training and education, as well as CSCW applications for jointly working with active objects. In contrast to discrete media, a continuous medium can change its state not only in response to user initiated operations, but also because of the passing of time. For this class of media the issue of consistency is still unexplored to a large extent. Only in the area of distributed virtual environments (DVEs) [12,13,5] have significant efforts been made to tackle the problem of consistency in replicated continuous interactive media.

As we shall show in this paper, the approaches for discrete media are not usable in the continuous domain. Moreover, the algorithms used for DVEs focus on optimizing responsiveness at the cost of frequent short-term inconsistencies. While this might be reasonable for a battlefield scenario, it provides only mediocre results for other continuous media. It is therefore necessary to investigate the issue of consistency in the continuous domain in a more general way than this has been done before.

In order to get a broader insight into consistency for the continuous domain, it is necessary to establish a common view on what consistency comprises for continuous media. Also, restrictions and tradeoff relationships need to be identified. Only after this has been done is it possible to evaluate the approach currently used by most DVEs and propose improvements. An essential part of this work is therefore dedicated to the formal specification of a consistency criterion for the continuous domain.

With this definition in place, we are then able to identify and to evaluate the tradeoff between responsiveness and short-term inconsistencies. This leads to the concept of local lag - deliberately decreasing responsiveness to lower the number of short-term inconsistencies. As a proof that local lag can improve the consistency related fidelity of replicated continuous interactive media, we have successfully applied the concept to an application which is able to share arbitrary active VRML (Virtual Reality Modeling Language) objects between a group of users. Throughout this work we use the terms continuous domain and continuous media as an abbreviation for replicated continuous interactive media, while the terms discrete domain and discrete

media are used synonymously for replicated discrete interactive media.

In Section Two of this paper we discuss the consistency model for discrete media and show why approaches which implement this consistency model fail in the continuous domain. The third section is dedicated to the formal definition of the term consistency for continuous media. Here we also examine the tradeoff between responsiveness and short-term inconsistencies. With these tools in place, a commonly used approach to achieve consistency in DVEs is evaluated in Section Four. This approach is called dead reckoning. A concept to exploit the tradeoff between responsiveness and short-term inconsistencies is proposed in the fifth section - it is called local lag. Section Six briefly outlines a 3D telecooperation application which makes use of local lag to achieve consistency. A summary and an outlook to future work is given in Section Seven.

CONSISTENCY IN REPLICATED DISCRETE INTERACTIVE MEDIA

A replicated discrete interactive medium is a medium that changes its state solely because of (user initiated) operations. Its state is kept in a replicated fashion for each participant of a session. Therefore each user initiated operation needs to be executed at each site.

Generally speaking, consistency for replicated discrete interactive media entails finding a 'correct' sequence of all the operations issued by the participants of a session and making sure that, at all participating sites, the state of the medium looks as if all operations had been executed successfully in that particular sequential order. To be more precise we use the terms causal ordering relation and dependent operation from Sun and Ellis [15,3,14], which are derived from Lamports work on virtual clocks [7]:

"***Definition 1**: Causal ordering relation "$\rightarrow$". Given two operations O_a and O_b generated at sites i and j, then $O_a \rightarrow O_b$, iff: (1) i=j and the generation of O_a happened before the generation of O_b, or (2) i≠j and the execution of O_a at site j happened before the generation of O_b, or (3) there exists an operation O_x, such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$.*

***Definition 2**: Dependent and independent operations. Given any two operations O_a and O_b. (1) O_b is dependent on O_a iff $O_a \rightarrow O_b$. (2) O_a and O_b are independent (or concurrent), expressed as $O_a || O_b$, iff neither $O_a \rightarrow O_b$, nor $O_b \rightarrow O_a$."*[15]

In order to illustrate these terms consider the session depicted in Figure 1: in this session the three operations O_1, O_2 and O_3 take place. The causal ordering relation for this example is $O_2 \rightarrow O_3$, since O_2 is executed at site 2 before O_3 is generated. An example for the semantics of O_2 could be that the user at site 1 marks a certain part of a text in a group text editor. In reaction to this, the user at site 2 chooses to delete a different section as operation O_3 (maybe because the section marked by the user at site 1 contained all the information of the deleted section). From the ordering relation it can be derived that O_3 depends on O_2, while O_1 and O_2, as well as O_1 and O_3, are independent of each other.

Based on the definition of the causal ordering relation, there are two consistency correctness criteria that are generally used to define the term consistency for replicated discrete interactive media [15,3,14]:

Based on the definition of the causal ordering relation, Ellis and Gibbs give the following consistency correctness criteria in the context of the GROVE (GRoup Outline Viewing Editor) system [3]:

(1) **Convergence property**. After all operations have been executed the state of the discrete interactive medium is identical at all sites.

(2) **Precedence property**. For any two operations O_a and O_b, if $O_a \rightarrow O_b$, then at each site O_a will be executed before O_b.

The first criterion ensures that all users will eventually see the same state of the interactive medium after all operations are completed, it is the fundamental consistency criterion. A violation of this criterion would result in different states of the replicated medium and therefore render it useless.

The second criterion makes sure that dependent operations are executed in order. If (2) were not required, the user at site 0 might see the result of dependent action O_3 before the result of action O_2 was visible. This would be confusing to the user since the causality of the operations would not be preserved.

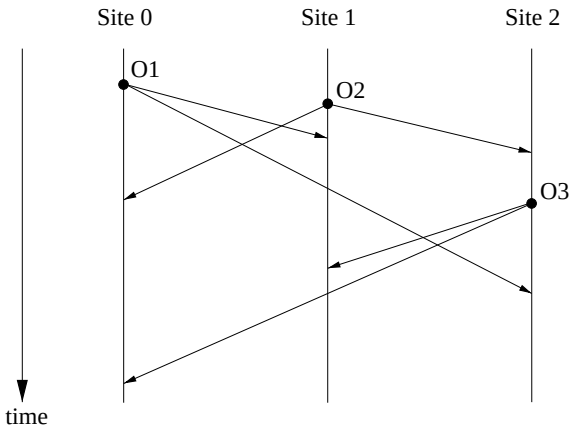

Figure 1: Discrete replicated interactive medium

In order to ensure that the consistency correctness criteria are not violated a number of different approaches exists. These range from a strict single user floor control policy, to more sophisticated algorithms such as operational transformation [3]. Since at least the convergence property is also desirable for replicated continuous interactive media, the question arises whether the approaches that ensure this criterion for discrete media could be reused in the continuous domain. Unfortunately the answer to this question is "No".

The reason why the approaches for discrete media fail when they are applied to continuous media is that consistency in replicated continuous interactive media is not only about finding a correct sequence of operations and making sure that at each site the result of all operations looks as if the operations had been executed in that sequence. In addition it requires that the result of all operations looks as if the

operations had been executed at the *correct point in time*. The algorithms for establishing consistency in the discrete domain can therefore be regarded as insufficient for the continuous domain.

To illustrate this problem let us examine a very simple example, based on a session involving a discrete replicated interactive medium. This session is attended by two users U_a and U_b. Now let U_a perform an operation on the medium. This operation will be executed first on U_a's copy of the interactive medium and some time later (because of the network transmission delay) on U_b's copy of the medium. Since for a discrete interactive medium a single operation cannot result in inconsistencies no consistency algorithms from the discrete domain will take any special actions in this example.

Now let us transfer this example to the continuous domain: Imagine a distributed simulation of a train that is approaching a switch. As in the discrete case this simulation has a replicated distribution architecture, so that each participant holds a local copy of the simulation's state. The simulation is attended by at least two spatially separated users, U_a and U_b. Operations caused by one of the participating users are broadcasted to all other simulation participants so that they can adjust the state of their copy of the simulation.

Just before the train arrives at the junction, U_a operates the switch. In the copy of U_a's simulation the operation takes place immediately. However, the information about U_a operating the switch will arrive at the copy of U_b's simulation at a later point in time. Applying the operation at this point in time to U_b's copy could lead to an inconsistent state because the train might have already passed the switch in U_b's copy of the simulation. As explained above, methods for ensuring consistency in discrete media will take no actions to correct this problem. This reveals the core reason why the mechanisms for consistency used in the discrete domain are not sufficient for continuous media, namely because they neglect the problem of executing operations at the correct point in time.

CONSISTENCY IN REPLICATED CONTINUOUS INTERACTIVE MEDIA

A replicated continuous interactive medium is a replicated medium that changes its state in response to (user initiated) operations as well as because of the passage of time. This definition implies that the medium has access to a physical clock that can be used to measure the progress of time. In the scope of this work we assume that the physical clocks of all the participants are reasonably synchronized - with a deviation of less than about 50ms (achievable using NTP [10] or GPS clocks). Furthermore we require that the correction of clock drift is done in a way that does not result in decreasing the value of a physical clock, e.g., it can be done by slowing down/halting the physical clock for a period of time instead of simply decreasing its value. If a given physical clock cannot be controlled in this way, some software may be required that adapts the reading of the physical clock so that it obeys this restriction. In the following we assume that this requirement is met.

In replicated continuous interactive media a user-initiated operation needs to be executed at a specific point in time, denoted by its timestamp. Similar to consistency in the discrete domain, consistency for continuous media is about finding a correct order of all operations. However, in addition it must be guaranteed that at all sites, the state of the interactive medium after these operations have been executed is the same as if the operations had been executed in the correct order *at the time identified by their timestamps*.

In order to be able to discuss this in a more formal way we define a partial ordering on the user-initiated operations as follows:

Definition 3: Partial physical time ordering relation "<". Given two operations O_a and O_b with timestamps T_a and T_b, then O_a is said to happen before O_b, expressed as $O_a<O_b$, iff $T_a<T_b$.

Definition 4: Any two operations O_a and O_b with timestamps T_a and T_b are said to be simultaneous, expressed as $O_a \approx O_b$, iff $T_a=T_b$.

Note that this definition is based on physical clocks, as opposed to logical-clock based methods for discrete media. The partial physical time ordering relation can be extended to become a complete physical time ordering relation by using an arbitrary tiebreaker for simultaneous operations. An example of such a tiebreaker could be the IP address, to break ties for simultaneous operations from two different participants, in combination with a counter, to break ties of the same participant.

Definition 5: Complete physical time ordering relation "<<". Given two operations O_a and O_b with timestamps T_a and T_b and tiebreakers B_a and B_b, then $O_a<<O_b$, iff (1) $T_a < T_b$, or (2) $T_a = T_b$ and $B_a<B_b$.

Now we are able to define the consistency correctness criterion for replicated continuous interactive media as follows:

(3) **Consistency Criterion for Replicated Continuous Interactive Media**. A replicated continuous interactive medium is *consistent* if after all operations have been executed at all sites, the state of the medium at all sites is identical to the state which would have been reached by executing all operations in the order given by the complete physical time ordering relation at the physical time denoted by the timestamps of the operations.

It is noteworthy that a precedence property is not part of the correctness criterion for replicated continuous interactive media. As we shall see later, the precedence property will be part of the fidelity criteria for the assessment of algorithms that realize the consistency correctness criterion.

The consistency criterion for the continuous domain is illustrated in Figure 2. The physical clocks of all three sites are slightly out of synchronization, the dashed lines identify the points in time with the same reading of the physical clock at each site. The small filled circles indicate the time when an action is executed. In the example the following ordering of operations applies: $O1<O2$, $O1<O3$ and $O2 \approx O3$ as well as $O1<<O2$, $O1<<O3$ and $O2<<O3$ (if the site number is used as the tiebreaker).

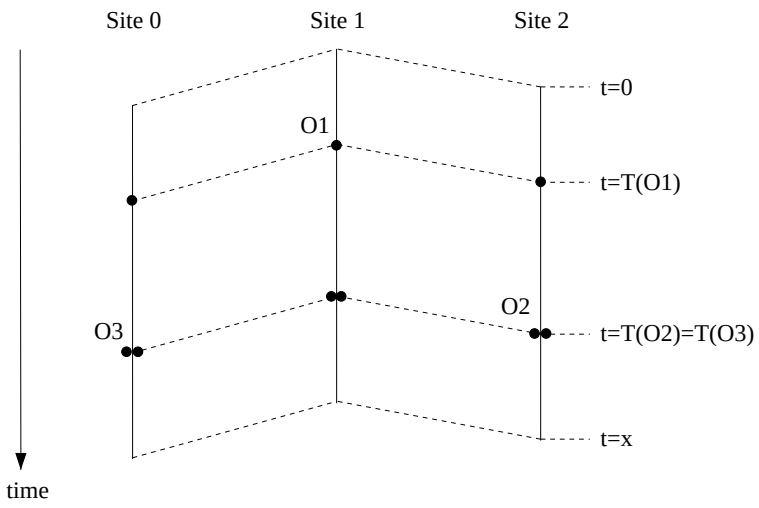

Figure 2: A consistent replicated continuous interactive medium

Figure 2 shows a somewhat unrealistic situation in which all operations are always known at all sites at the time denoted by their timestamp. This situation, however, is important, since the meaning of the consistency criterion is that at some time after the last site has received all operations (for example at t=x), the medium will look as if the operations had been executed as depicted in Figure 2.

By guaranteeing that the state of the medium will be identical at all sites after all operations have been resolved, the consistency criterion is the most important prerequisite to making sure that a replicated continuous interactive medium is actually usable. However, even if the consistency criterion is guaranteed by a proper algorithm, it is still possible that consistency-related, transient artifacts occur. In addition to the basic consistency criterion we therefore define two additional fidelity criteria:

(4) **Avoidance of short-term inconsistencies**. If an operation with timestamp T is not executed at site i at the physical time T, then this operation is said to have caused a short-term inconsistency of the interactive medium at site i. Ideally, short-term inconsistencies should be non-existent.

Short-term inconsistencies occur when an operation issued at site j arrives at site i after the time denoted in the timestamp of the operation. Because we assume that an algorithm exists to ensure the consistency criterion (3) a late arrival means that the state of the interactive medium at site i needs to be repaired. In the train example this would mean moving the train from the wrong position on one branch of the tracks to the right position on the other branch of the tracks. The fidelity criterion for the avoidance of short-term inconsistencies includes the meaning of the precedence property (2) as it has been specified for discrete media since a reversal of the precedence of operations can only be caused by the late arrival of remote operations.

Short-term inconsistencies basically have three implications: They may cause dependent operations to be executed in the wrong order, they usually result in visual artifacts (the train “jumps” from one position to a different one), and a user might take actions that are based on an inconsistent state of the medium. An example for the latter is the following situation: Assume that in the train example U_b pulls the brakes of the train after it has passed the switch but before the operation from U_a has arrived. In this case, U_b intends to stop the train because it is going in the wrong direction. After U_b has issued this operation, the operation from U_a arrives (late). In order to satisfy the consistency criterion (3), the train will jump to the correct position. The net result of both operations is that the train will stop while it is headed in the right direction, which is a clear violation of the intention of the users.

(5) **Low response time**. The response time of an interactive medium is the time between the physical time a user issues an operation and the timestamp of that operation. Assuming that local processing time, memory access, and rendering requires only negligible time, the ideal response time should be zero.

If the response time exceeds a certain threshold the users will notice that a delay exists between the time the operation was issued and the time when the operation is executed. This will result in an ‘unnatural’ behavior of the medium. Response time and *responsiveness* are used synonymously in the context of this work.

While it would be desirable to be able to guarantee that no consistency-related artifacts occur in a replicated continuous interactive medium, this is not possible. The reason for this is that the optimization of the response time and the avoidance of short-term inconsistencies are conflicting goals.

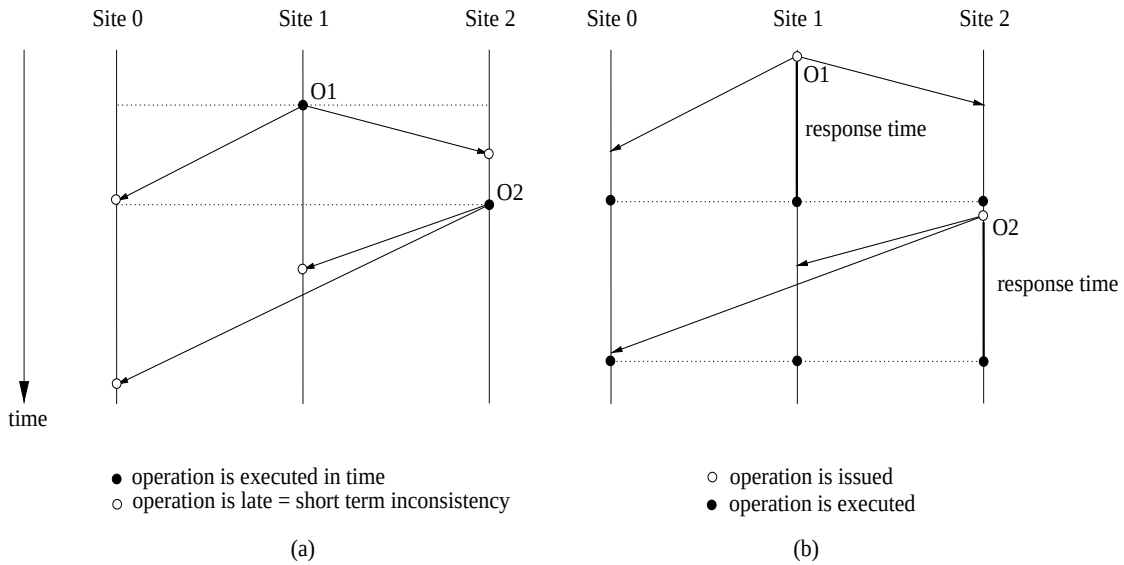

Figure 3: Short-term inconsistency vs. response time: the trade-off

Part (a) of Figure 3 shows an example where the responsiveness is optimal (response time = 0). In this case the operations issued by the users take effect immediately. The points in time when O_1 and O_2 should take effect are indicated by the dotted line. As can be seen, only the user issuing the operation does not experience short-term inconsistencies. Any other user will have a short-term inconsistency for that operation due to the transmission delay of the remote operation.

Part (b) of Figure 3 shows the same situation optimized for the short-term inconsistency criterion. Here short-term inconsistencies are avoided at the cost of increasing the response time for each operation to the maximum transmission delay between any two participants. Note, that the execution of O_1 and O_2 is delayed at the originating sites until all participants have a chance to act simultaneously. An event received at a time before it should be executed does not pose a problem. It should be buffered by the receiver until the time denoted by its timestamp is reached.

An observant reader might have noticed two peculiarities:

- What if the time deviation between sender and receiver happens to compensate for the transmission delay?
- In an environment where network packets might get lost, how exactly can we calculate the maximum transmission delay for an operation?

The answer to the first question is that the time deviation might indeed compensate for the network delay between a given sender and receiver. This makes it possible for the sender to have a response time equal to zero while the receiver does not experience short-term inconsistencies.

However, further reflection reveals that this works only when the sender and the receiver do not switch their roles. As soon as the previous receiver becomes the sender short-term inconsistencies will occur because of the large time deviation which now prevents that any operation arrives at the new receiver in time.

The second observation arises from the problem that in an environment where packet loss occurs, it is impossible to define an upper bound on the delay which an operation needs to arrive at a remote site. After all, the same packet might get lost over and over again as it is being retransmitted by the sender. This leads us to the conclusion that, while a reduction of short-term inconsistencies is desirable, a guaranteed prevention is not possible in real networks.

With the definition of the convergence property in place and the tradeoff between the two fidelity criteria discussed, we will now investigate a frequently used algorithm for achieving consistency in replicated continuous interactive media.

DEAD RECKONING

An approach that is commonly used to guarantee that the consistency criterion is satisfied in distributed virtual environments (e.g., multi-user virtual reality and battlefield simulations) is a combination of state prediction and state transmission. In this approach the application “knows” how the objects in the virtual environment should behave over time. Examples are a plane that will fly straight at constant speed or a projectile that will take a course dictated by the physical law of gravity. The ability to predict the behavior of objects is called *dead reckoning.*

Each object for which dead reckoning is performed has a single controlling application, e. g., for a plane this will be the application of the pilot. The controlling application is responsible to inform peer applications when the state of the object deviates by more than a certain threshold from the predicted state. In the event of a deviation, the controlling application transmits the complete state of the affected object to all peers. Upon receiving this information, an application discards the outdated state and uses the new state to perform dead reckoning for this object. In order to

be able to use this approach over unreliable transport services, the controller of an object additionally transmits the state of the object at regular intervals in the form of so called heart-beat messages.

User initiated operations are immediately applied to the local state of the affected object. The object will thus be put into a state that differs significantly from the predicted state, thereby requiring the controlling application to transmit the new state to its peer applications. Because of this behavior the dead reckoning approach optimizes the response time criterion, while each action will result in a short-term inconsistency at each remote site.

While this approach has a number of important advantages such as robustness and scalability, its disadvantages limit its applicability:

- It causes the maximum number of short-term inconsistencies.
- It requires that the state of each object be transmitted for each operation. This could lead to a massive need of bandwidth, especially when the size of the state of the object exceeds a couple of bytes.
- Only one controller for each object is possible. This prevents collaborative actions, such as two users moving a single object.

An example where this approach is used, is the well known battlefield simulation protocol called DIS (Distributed Interactive Simulations) [13] and general purpose DVEs such as DIVE [5]. In a typical battlefield simulation the state of the relevant objects is small (position and velocity) in size and relatively easy to predict, while each object will have only one controller (e.g., the pilot of a plane, the driver of a car, etc.). The number of visible artifacts can be reduced by using intelligent algorithms to repair the state instead of immediately adopting the new state. For example, after a plane has changed its direction a remote application can calculate a alternate flight path which will eventually bring the plane back into the correct state (rather than jumping immediately to the correct position). Because of these attributes the state prediction and transmission approach is adequate for battlefield simulations and some distributed virtual environments.

However, this approach becomes less appropriate if an interactive medium has one or more of the following properties:

- the state of objects is complex,
- visual artifacts cannot be concealed,
- true collaboration is important, or
- actions based on an inconsistent state are critical.

For several applications it might therefore be better to have a closer look at the tradeoff between responsiveness and short-term inconsistencies before simply maximizing responsiveness. In the following section we will discuss how to exploit the tradeoff to improve the consistency-related fidelity of replicated continuous interactive media.

LOCAL LAG

The concept of *local lag* is simple: Instead of immediately executing an operation issued by a local user, the operation is delayed for a certain amount of time before it is executed. To use the terms from section 3, this means that the value of an operation's timestamp will be greater than the point in time when the operation is issued by the user. The delay that is introduced for the local user in this way is called local lag. As depicted in Figure 3 (b), if the value for local lag is sufficiently high, then it can reduce the number of short-term inconsistencies.

The concept of local lag is related to the work of Cristian et al. [2] where a total ordering on broadcast messages from multiple senders is established to achieve atomic broadcast. In this approach operations also get a timestamp which is greater than the point in time the operation is issued by a sender. Operations are executed at the time denoted by that timestamp. In order to calculate the timestamp Cristian et al. assume that an upper bound on the transmission delay can be given, so that operations never arrive after a certain time.

In contrast we assume that such an upper bound does not exist and that we have to cope with operations that arrive late. The local lag approach therefore differs in two main aspects:

- The amount of local lag is calculated in a different way. It is adapted to the specific needs of consistency for replicated continuous interactive media.
- There exist repair mechanisms should an operation arrive late.

Determining a Value for Local Lag

Clearly it would not be desirable to move from one extreme, where the response time is zero but short inconsistencies are very frequent, to the opposite extreme, where almost no short-term inconsistencies occur but the response time is unacceptably high. Therefore it is important to consider both consistency related fidelity criteria. For a given replicated continuous interactive medium we propose to do this in three steps: (1) determine a minimum for the local lag needed to prevent a significant amount of short-term inconsistencies, then (2) determine the highest acceptable response time, and finally (3) choose a value for the local lag.

Step 1*. Determining a minimal value for local lag.*

In order for local lag to be useful it needs to significantly reduce the number of short-term inconsistencies for all participants. Moreover, short-term inconsistencies should be reduced for each sender/receiver pair. We therefore propose to use the maximum of the average network delays between any two participants as the minimum amount of local lag. Typical values for network delays (assuming an uncongested network) are: less than 1ms for a LAN, 20 ms within a European country, 40 ms within a continent, and 150 ms for a world wide session. Choosing the maximum of the average network layer delays as minimal value for local lag implies that short-term inconsistencies will occur only if packets get lost or the jitter becomes significant because of network congestion.

If the clocks of the participants are not completely synchronized, the maximum offset between any two clocks needs to be added to the result. This is necessary because the

sender of an operation might be behind in time relative to the recipient of the operation. If this time deviation were not accounted for, the time deviation plus the transmission delay might be larger than the local lag which would result in a short-term inconsistency. This problem is shown in Figure 4 where site 0 is ahead in time of site 1. Even though the operation O1 from the user at site 1 has a local lag greater than the maximum network delay, a short-term inconsistency occurs at site 0 due to the deviating clocks. Typical values for time deviation of computers using NTP or SNTP are 20-50 ms if the NTP server is reached via WAN and less than 10 ms if the NTP server is part of the same LAN.

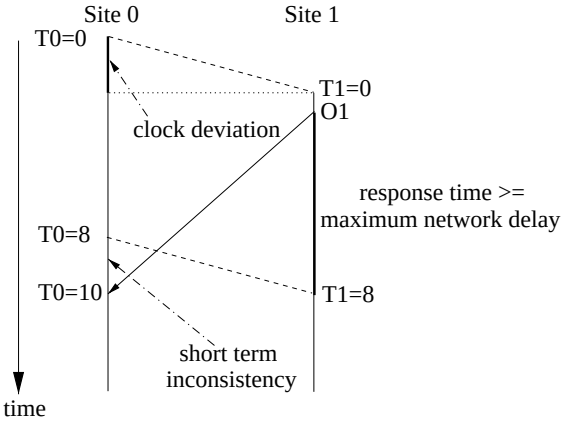

Figure 4: Problem with deviating clocks

***Step 2**. Determining the highest acceptable response time*

The maximum local lag is dependent on how much response time a user can tolerate for a given interactive medium. In order to determine this value, the work conducted in the area of System Response Time can provide a good orientation [16,11,1]. Furthermore, for a given specific medium it may be a good idea to conduct perceptual psychological experiments. Ideally the experiments will deliver two values: the first is the amount of local lag for which a user cannot notice the delay. The second value is the amount of local lag which can be tolerated by the user, i.e., the delay can be noticed but it is not disturbing. It seems to be reasonable to use the second value as the maximum amount of local lag for an interactive medium.

The literature about System Response Time and first experiments we have conducted for specific replicated continuous interactive media using local lag suggest that a lag value of 80-100 ms is not noticeable by the user, independent of the operation and the medium. The value at which local lag gets to be disturbing seems to depend heavily on the operation, ranging from 300-400 ms for simple click operations to 100-150 ms for drag operations.

***Step 3**. Choosing a value for the local lag.*

The previous two steps can result in two main cases: either the minimum value for local lag is smaller than or equal to the highest acceptable response time, or it is not. In the first case a value between the minimum value for local lag and the highest acceptable response time can be chosen. This choice should be based on additional criteria such as whether additional time is required to receive forward error correction packets, etc.

If the highest acceptable response time is lower than the minimal useful value for local lag, then a true tradeoff situation occurs. Given the values mentioned above this should be relatively rare, though it might happen for very demanding media and applications. In this case it is necessary to lower the fidelity of both criteria in a way which provides the best overall fidelity to the user. Most likely this will require another set of perceptual psychological experiments.

Repairing Short-Term Inconsistencies

If the amount of local lag is chosen well, it eliminates a significant amount of short-term inconsistencies. However, it does not completely prevent all inconsistencies since operations might still arrive late or not at all (e.g., if the network is congested or a transport packet gets lost). It is therefore necessary to have a mechanism that will repair the state in these cases. In the following we summarize three possible approaches to repairing inconsistent states.

State prediction and Transmission.

An obvious approach is to combine local lag with the mechanism for state prediction and transmission. Instead of immediately executing a local operation on an object, the operation is delayed using the value for local lag: the operation gets a timestamp equal to the current time plus the value for the local lag. However, when the user issues the operation, the new state for the object at the time denoted by the timestamp of the operation is calculated immediately. This state includes the operation and bears the same timestamp as the operation. The calculated state is immediately transmitted to all participating sites.

When the physical time identified by the timestamp is reached, the site where the operation was issued will execute the operation. The remote sites that by then have received the new state for the object will start using the new state for dead reckoning without ever encountering short-term inconsistencies.

A site that receives the new state late, e.g., because of an unexpected transmission delay, experiences a short-term inconsistency. This inconsistency is repaired just as for regular dead reckoning by replacing the local copy of the object's state with the state that was transmitted by the controller of the object.

The combination of local lag with state prediction and transmission retains two limitations from the original approach without local lag: only one controller per object is possible and the state of an object must be small (since the state is still transmitted for each operation). In the following approach we will allow the state of objects to be complex.

Requesting States

In order to allow objects to have a complex state of a size significantly larger than the 200 - 300 bytes used for states in the DIS protocol, it is important that the state transmissions be restricted to cases when a short-term inconsistency needs to be repaired. In all other cases only the operation itself will be transmitted.

In this approach a recipient of a transmitted operation checks the timestamp of the operation. If the operation has arrived in time, it will be buffered until it is due for execution. Only if the operation is late will the state be requested to repair the short-term inconsistency.

An algorithm realizing this approach needs to be able to identify those participants that have the correct state of the object. In addition it must be guaranteed that at least one participant can determine the correct state of the object. A possible solution to these problems is to allow only a single controller at a time for each object. Since only the controller is able to issue operations for the object, the controller of an object will always have the correct state. It is the task of the controller to reply to requests for the state of the object. The controller of an object can change, as long as it is guaranteed that all operations (and the repair of short-term inconsistencies induced by these operations) of the old controller have been executed before a different participant becomes the new controller of the object.

The main limitation to this approach is that no true collaborative operations are possible, i.e., two users are not able to interact with the same object at the same time (unless some more sophisticated algorithm is used to determine who will reply to a state request). Also, the repair of an inconsistent state requires an additional roundtrip for the state request/reply. Therefore the requesting states approach is useful primarily for applications where simultaneous access to a single object is not required and the likelihood of an inconsistent state is rather low.

Time Warp

This method is an adoption of the time warp algorithm used for optimistic parallel discrete event simulation [4,6]. In time warp the state of each object does not need to be transmitted, even when a short-term inconsistency occurs. Instead local information is used to repair the state. Additionally time warp allows an arbitrary number of users to interact with the same object at the same time.

The fundamental idea of time warp is that each participant saves the state of the interactive medium at certain times. All operations up to a certain point in time in the past are kept in a log. When a short-term inconsistency occurs, the interactive medium is rolled back to the last saved state of the interactive medium before the operation should have taken place. Then the operation that caused the short-term inconsistency is inserted into the log. After that the medium is played in fast forward ("time-warped") mode, executing the operations from the log at appropriate times until the current time for the medium is reached, and operation is resumed at the normal pace. Only the end result of this operation should be visible to the user.

Choosing the points at which to save the state depends on the application and the medium. If the state is saved too frequently, it might effect the visual quality of the presentation and use a large amount of memory. If the state is not saved frequently enough, many operations will have to be performed when a short-term inconsistency occurs, extending the time during which the inconsistency is visible to the user.

The main drawback to this approach is that it requires a sophisticated application that is able to handle the process of time-warping. In addition, if the state is complex, then the time warp approach might consume a large amount of memory to save the states, as well as place a heavy load on the computer when a time warp has to be executed.

In summary it can be said that the choice of the amount of local lag as well as choosing a repair mechanism for short-term inconsistency depend heavily on the application and the medium itself. In the following section we will show how these decisions have been made for a 3D telecooperation application.

USING LOCAL LAG IN A 3D TELECOOPERATION APPLICATION

We have tested the concept of local lag in the context of the 3D telecooperation application TeCo3D - a shared workspace for dynamic and interactive 3D models [8]. TeCo3D was developed to allow users to share collaboration-unaware VRML (Virtual Reality Modeling Language) models, i.e., models which have not been specifically developed to be used by more than one user at a time. With this functionality it is possible to include arbitrary VRML content, as generated by standard CAD or animation software, into teleconferencing sessions.

TeCo3D was developed by reusing the Java3D VRML loader [17] as 3D presentation and execution engine and employs a completely replicated distribution architecture with reliable multicast as means of communication. When a user imports a local VRML object, the VRML code is parsed and the parts which are responsible for user interactions are replaced with custom components turning the collaboration-unaware object into a collaboration-aware model. User initiated operations are captured by the custom components and are transmitted to peer instances in the session, where they are injected into the local model. In order to provide remote users with the initial state of the imported objects and to repair short-term inconsistencies we have enhanced the VRML loader by a method to get and set the state of arbitrary VRML objects [9].

The algorithm which provides consistency for TeCo3D had to take into consideration that the state of VRML objects can be complex, ranging from 200 bytes for a simple moving sphere to 50,000+ bytes for an interactive cartoon. Extracting and setting the state of objects is costly, both in terms of visual artifacts (the animation freezes for 100-200 ms) and computational overhead. It was therefore of critical importance to minimize the number of short-term inconsistencies. The minimal useful amount of local lag (step 1) was therefore determined as 200 ms = 150 ms (world-wide conferences) + 50 ms (maximum time deviation).

First tests with increasing the response time from zero up to a value of 300 ms have shown that a delay of 100 ms is barely noticeable and 200 ms is still very acceptable for most interactive 3D objects. In addition users seem to be able to adapt to a constant delay rather fast, making a response time of up to 300 ms feasible for most types of interaction. The maximum acceptable response time (step 2) was therefore estimated to be between 200 and 300ms.

With a minimum value for the local lag at 200ms and the maximum value between 200 and 300 ms we have chosen to set the local lag to 250ms, adding 50 ms to the minimum value for some additional protection against network jitter.

The next decision which had to be made was the choice of the repair mechanism. The state prediction approach was not taken into consideration because of the complexity of the state information for VRML objects. While it was tempting to use time warp, we did not want to burden our prototype with the complex mechanisms necessary for this approach. We therefore chose the requesting states method to repair short-term inconsistencies. The implementation of both local lag and repair mechanism were straight forward and needed less than 500 lines of code.

First experiments with TeCo3D show that, as expected, no short-term inconsistencies occur as long as all network packets arrive at their destination. When packet loss increases (and therefore retransmitted operations arrive late), short-term inconsistencies start to appear. In our test sessions with loss rates between 0% and 5% these cases where sufficiently rare to not significantly affect the consistency related fidelity of the application.

In the future we will use forward error correction to protect user initiated operations from packet loss, effectively giving the transmission of operations a higher priority than the transmission of states for repairing short-term inconsistencies. This should significantly increase the maximum tolerable packet loss rate.

CONCLUSION AND OUTLOOK

In this paper we investigated the problem of consistency in replicated continuous interactive media. It was shown that algorithms for consistency in the continuous domain differ significantly from those used for discrete media. In order to systematically approach the problem, a formal definition of the term consistency in the continuous domain was deduced from the special characteristics of the continuous media class. Based on this definition an important tradeoff relationship between responsiveness and the occurrence of short-term inconsistencies was examined. We proposed to make use of the knowledge of this tradeoff relationship in order to increase the consistency related fidelity of the medium. This can be done by deliberately increasing the response time in order to decrease the number of short-term inconsistencies, leading to the concept of local lag. The local lag approach for consistency in the continuous domain was proven to be usable by integrating it into a 3D telecooperation application.

Currently we focus our work on a more thorough investigation of the repair mechanisms used in combination with local lag. It is especially tempting to employ the time warp approach for our 3D telecooperation application. However, before we are able to use time warp as an alternative to requesting states, we need to make sure that the complex algorithm of time warping and the overhead for regular state saves does not limit the fidelity of the application.

ACKNOWLEDGMENTS

This work was supported through a Ph.D. grant by SIEMENS. I wish to thank my colleagues Volker Hilt, Christoph Kuhmünch, and Jürgen Vogel for many important discussions about local lag. Furthermore the anonymous reviewers provided very helpful and knowledgeable comments that helped me to improve the paper.

REFERENCES

1. S. Card, T. Moran and A. Newell. The psychology of human-computer interaction. Hillsdale, NJ, Lawrence Erlbaum Associates, 1983.
2. F. Cristian, H. Ahali, R. Stron, and D. Dolev. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. *In Proc. of the 15th Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 200 - 206, IEEE Computer Society Press, 1985.
3. C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In: *Proceedings of the 1989 ACM SIGMOD Conference on Managements of Data*, Portland, OR, USA, 1989, pp. 399 - 407.
4. R. M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, Volume 33, Number 10, 1990, pp. 30 - 53.
5. E. Frécon and M. Stenius. Dive: A scaleable network achitecture for distributed virtual environments. *Distributed Systems Engineering Journal* (special issue on Distributed Virtual Environments), Vol. 5, No. 3, 1998, pp. 91 - 100.
6. D. R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, Volume 7, Number 3, 1989, pp. 404 - 425.
7. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, Volume 21, Number 7, 1978, pp. 558 - 565.
8. M. Mauve. TeCo3D: a 3D telecooperation application based on VRML and Java. In: *Proc. of SPIE Multimedia Computing and Networking (MMCN) '99*, San Jose, CA, USA, published by SPIE, Bellingham, Washington, USA, January 1999, pp. 240 - 251.
9. M. Mauve. Transparent Access to and Encoding of VRML State Information. In: *Proc. of the Fourth Symposium on the Virtual Reality Modeling Language (VRML) '99*, Paderborn, Germany, published by ACM SIGGRAPH, New York, USA, February 1999, pp. 29 - 38.
10. D. L. Mills. *Network Time Protocol (Version 3) specification, implementation and analysis*. DARPA Network Working Group Report RFC-1305, University of Delaware, 1992.
11. B. Schneiderman. Response Time and Display Rate in Human Performance with Computers. In: *ACM Computing Surveys*, Vol. 16, No. 3, 1984.
12. S. K. Singhal. *Effective Remote Modeling in Large Scale Distributed Simulation and Visualization Environments*. Ph.D Dissertation, Department of computer-science, Stanford University, 1996.

13. S. Srinivasan. Efficient Data Consistency in HLA/DIS++, in: *Proceedings of the 1996 Winter Simulation Conference*, 1996, pp. 946-951.

14. C. Sun, X. Jia, Y. Zhang, Y. Yang and D. Chen. Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interactions*, Vol. 5, No. 1, March 1998, pp 63 - 108.

15. C. Sun and C. Ellis. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements. In: *Proceedings of the ACM 1998 conference on Computer Supported Cooperative Work (CSCW'98)*, Seattle, Washinton, USA, 1998, pp. 59 - 68.

16. S. Teal and A. Rudnicky. A Performance Model of System Delay and User Strategy Selection. In: *Proc. of Human factors in computing systems 1992*. Monterey, CA, USA, 1992, pp. 295-305.

17. VRML Consortium. The Java3D and VRML Working Group. Web page: www.vrml.org/WorkingGroups/vrml-java3d/.