2008 14th IEEE International Conference on Parallel and Distributed Systems

Measures for Inconsistency in Distributed Virtual Environments

Sven Grottke*
Institute of IT Services
University of Stuttgart
sfs@cs.tu-berlin.de

Jan Sablatnig*, Jiehua Chen*, Ruedi Seiler
Institute of Mathematics
Technical University of Berlin
{jon,chen,seiler}@math.tu-berlin.de

Andreas Köpke*, Adam Wolisz
Institute of Telecommunication Networks
Technical University of Berlin
koepke@tkn.tu-berlin.de, awo@ieee.org

Abstract

In distributed virtual environments, hosts typically have to react to events within a time span which is less than the network latency. As a consequence, hosts do routinely take actions although the system is in an inconsistent state. This has a noticeable influence on the perceived quality of these actions and their effect on the application. We argue that the level of this influence depends on the degree of inconsistency. In this paper, we tackle two fundamental questions: How does the degree of inconsistency influence the perceived quality of the users' actions? How can the degree of inconsistency be quantified? We propose a benchmark test for comparing different consistency algorithms with each other which consists of two measures of inconsistency and a sample scenario. For two different consistency algorithms, we compare the results of our benchmark test with the results of a user evaluation test and a simple yield measure.

1. Introduction

The number of distributed virtual environments (DVEs) in online gaming, education and other areas of application keeps increasing. They typically consist of between several dozens and up to tens of thousands of users with computers connected via the Internet. Each of them can manipulate objects within the DVE and interact with other users.

A core component of DVEs is the consistency algorithm, which ensures that all participants see the same world, and that changes induced at one host are propagated to all other hosts.

While some attempts at standardization exist, such as HLA or DIS, most commercial and research DVEs use a custom-made consistency algorithm. This leaves the potential developer of a new DVE with an interesting question: which consistency algorithm to choose for a specific application?

Clearly, answering this question requires the means to compare different algorithms with each other regarding the application at hand. Here, we focus on how the consistency algorithm influences the quality of the application as perceived by the users. To illustrate the problem for a typical application scenario, let us consider the case of a virtual soccer game to be played over the Internet by 22 participants, each controlling a single avatar representing a "player":

On the one hand, a player would be seriously annoyed if the ball would suddenly disappear from his foot or change trajectory in midair without any discernible reason. On the other hand, it would be equally annoying if a player kicks the ball and the ball wouldn't start moving immediately, but take off after a second or so.

As the network delay makes it impossible to keep the system responsive and perfectly consistent at the same time, users will routinely take actions while the system is in an inconsistent state. In this paper, we deal with the following questions: How does the degree of inconsistency influence the perceived quality of the users' actions? Which quantitative measures can be used to evaluate this degree of inconsistency?

*Jiehua Chen, Sven Grottke, Andreas Köpke and Jan Sablatnig were supported by a grant from the Deutsche Forschungsgemeinschaft (DFG).

1521-9097/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPADS.2008.87

2. Problem Description and Basic Concepts

To define the requirements on an inconsistency measure, we have to take a closer look at the typical characteristics of a distributed virtual environment. Such systems consist of a number of hosts which are connected via a network, e.g. the Internet or a set of radio modems. Communication is delayed and may be faulty.

The hosts communicate to work together on a common task, e.g. a virtual soccer game. To this end, they share a set of variables, such as the position of a player or ball in the virtual soccer game. Each host has its own idea of the value of each variable at any given time.

More formally, for a system with A shared variables, the vector $\vec{v_h}(t) = (v_{1,h}(t) \dots v_{A,h}(t))$ describes the state of the world as seen by host h at (wall-clock) time t. We call $\vec{v_h}(t)$ the *world view* of h at time t. We call the matrix $V(t)$ the state of the system at time t, with each column vector representing the world view of one host at time t. We call a system *consistent* at time t iff all hosts have the same world view at this time.

If an event takes place at a host, such as a player kicking the ball, this event has to be propagated to the other hosts to make the system consistent again. An ideal system, where communication is unhindered by delays or errors, would be consistent at any time. However, this is impossible in any real world distributed system, because the speed of light puts a lower limit on the time required to propagate events to other hosts. Furthermore, there is an upper limit of ~ 150 ms [1] on how long a host may wait before reacting to a *local* event, which is typically less than or equal to the network delay.

There are several different approaches for keeping a DVE consistent. For an in-depth discussion, which is out of the scope of this paper, we refer to the overviews given in [3] and [4]. The most popular approach is *loose consistency* as described in the DIS standard [7], followed by *optimistic consistency* as described by Mauve et al. [8]. In systems employing either approach, hosts react to local events before they have been propagated to all other hosts in the system. Both approaches differ in how they resolve conflicts between events. With the loose approach, messages do not describe events, but the sender's world view at send time. The receiver simply overwrites its current world view with data from the message. It does not attempt to reconstruct some chronological ordering of all messages. With the optimistic approach, messages describe local events at the sender. Each host keeps a history of events, and adds events from received messages to this history. When a late event arrives at a host, i.e. a newer event has already been received, the host integrates it into the history by backtracking the application to the time when the event occurred, and recomputing the history from this time.

As we've argued before, hosts in such systems will typically have to react to events and take decisions while the system is in an inconsistent state. This phenomenon is referred to as *short-term inconsistency* [8]. The impact of short-term inconsistency on the system is twofold:

1. Inconsistencies cause users to make improper decisions. Even small differences in the value of a single variable may have effects which are highly sensitive in the amount of the actual error. Imagine a penalty kick in the final of the virtual soccer world cup: if, when the goalkeeper reacts, his view of the position and direction of the ball differs even slightly from the kicker's view, he will decide to jump in the wrong direction, missing the ball and losing the cup.

2. Usually, users don't immediately notice that the system is inconsistent. Eventually, however, inconsistencies have to be resolved, typically when a host receives a message about an event from another host and updates its own world view accordingly. This causes non-continuous changes in the host's world view which users typically notice as unexpected, annoying effects. As an example, let us assume that the ball is flying in a ballistic curve towards a player. When a message is received that another player actually changed the direction of the ball 500 ms ago, this causes the player to see the ball suddenly changing direction and "jumping" to a new position in mid-air, destroying the illusion of a real-life soccer game.

If, for an application employing a specific approach to the consistency problem, we can quantify the inconsistency, then this would allow us to draw conclusions on the quality of this approach with regard to the application at hand. Furthermore, we also want an inconsistency measure to be applicable to a wide range of consistency algorithms and application scenarios, and to give results which can be used to compare different consistency algorithms with each other.

3. Related Work

A number of consistency models have been developed mainly originating from distributed databases, shared memory models and multiprocessor computing. Typically, they give guarantees about the ordering of read and/or write operations, such as in the linear, sequential or causal consistency models [5, 3]. They are not very well-suited to evaluating DVEs due to the continuous changes of variables inherent in a virtual environment. There have been attempts at maintaining causal relations between events in a DVE by Roberts and Sharkey [10], and by Mauve et al. [8]. However, all

these efforts have in common that they fail to measure the effects of short-term inconsistency.

Cronin et al. used the cost of rollback operations to compare the performance of different optimistic consistency algorithms [2]. Their methods only work for optimistic algorithms, and are not generally applicable to other approaches. Furthermore, they do not take the achieved degree of inconsistency into account.

Liang and Boustead counted the number of kills per second scored by robots playing Quake III over a network [6]. They found that the number of kills correlates well with the average network latency. This is similar to our proposed yield measure (ref. 4.3).

Palazzi et al. used game-time difference (GTD) to measure performance in a DVE [9]. GTD is defined by the difference between the time when an event arrives at a remote host and the time when it was generated, i.e. it measures the duration of short-term inconsistency for a single event. However, it ignores the degree of inconsistency and its influence on game play.

Zhou et al. defined a metric which they call *time-space inconsistency* [12]. It is defined as the difference between the local and remote positions of the most important object in a DVE. They also propose a method to estimate time-space consistency of a new DVE at design time. Their metric relies on the existence of a single most important object, which is not always the case for large-scale DVEs. Furthermore, their estimation method only works for loose consistency algorithms.

4. Proposed Inconsistency Measures

As we have shown, existing measures cannot be applied to the systems under investigation, or fail to capture the full range of effects of inconsistency on the application. Thus, we propose a number of new measures which are better suited to meet our requirements.

As we pointed out in section 2, an inconsistency measure has to take into account the different effects of inconsistency on the perceived quality of an application. Thus, a good measure has to take into account the effect of inconsistency between hosts each time a decision is being made by a user, and the degree of disturbance caused by non-continuous changes due to messages being received. As it is quite difficult to design a measure which captures both aspects, we propose several measures, each quantifying one relevant aspect of inconsistency. The divergence measure highlights the difference in the world views, the discontinuity measure emphasizes the graphical effects of the inconsistency and the yield measure stresses how well the users can play the game despite the inconsistency.

4.1. Divergence Measure

The divergence measure measures the average difference between the world views of all hosts.

Consider a system where A denotes the total number of variables, and N the total number of hosts. We define a $A \times N$ matrix V, where each column represents the world view of one particular host. We define a second $A \times N$ matrix R, $r_{a,h} \in [0,1]$. $r_{a,h}$ describes the interest host h takes in variable a. As both V and R change over time, we write $V(t)$ and $R(t)$ to describe the system state and interest matrix at time t. We define

$$d_a(t) = \sqrt{\sum_{h=1}^{N} \frac{r_{a,h}(t)}{\sum_{\hat{h}=1}^{N} r_{a,\hat{h}}(t)} \left(v_{a,h}(t) - \overline{v_a(t)}\right)^2} \quad (1)$$

as the divergence of the system at time t with regard to variable a, where

$$\overline{v_a}(t) = \sum_{h=1}^{N} \frac{r_{a,h}(t)}{\sum_{\hat{h}=1}^{N} r_{a,\hat{h}}(t)} v_{a,h}(t) \quad (2)$$

is the weighted average value of variable a over all hosts[1] at time t. In typical implementations, $r_{a,h}(t)$ will be either 1, or 0 if a host is not interested in a particular variable. Please note that expression 1 is a weighted standard deviation of the world views of all hosts. We define

$$d(t) = \sum_{a=1}^{A} \frac{w_a(t)}{\sum_{\hat{a}=1}^{A} w_{\hat{a}}(t)} d_a(t) \quad (3)$$

as the divergence of the system at time t. $W(t)$, with $w_a(t) \in \mathbb{R}$, denotes the relevance vector, i.e. the relevance of each variable with regard to the overall inconsistency. The relevance of a variable is chosen manually at design time. It is typically based on the perceived influence of this variable on the user decisions. For example, the exact position of the ball would be very relevant to the decisions of all soccer players on the field, while the color of the shirt of a particular spectator would be almost irrelevant.

For a system with a session length of T, we define the divergence of the system history as

$$D(T) = \frac{1}{T} \int_0^T d(t)dt \quad (4)$$

$D(T)$ is the value we use as the inconsistency measure.

The divergence measure is applicable if it is possible to determine the world views of all hosts at any given time. Its validity depends on the particular choice of w. The computational complexity scales with $O(N \cdot A \cdot T)$.

[1] We assume that $\forall t \forall a \exists h : r_{a,h}(t) > 0$, i.e. at any time t and for each variable a, there's at least one host which is interested in a at this time.

4.2. Discontinuity Measure

The discontinuity measure quantifies the disturbance caused by sudden changes in a host's world view which are due to variables being changed by messages from other hosts as described in section 2.

The discontinuity measure is measured separately for each host. Suppose a message m changes the value of variable a at host h and time t_m. Let V and V' denote the system state matrix with and without receiving the message. The change m causes in h's world view is expressed by

$$g_{a,h}(m) = (v'_{a,h}(t_m) - v_{a,h}(t_m)) \tag{5}$$

Let M_h be the the set of all messages received by h during one session. The average discontinuity at h is written as

$$G_h = \frac{1}{T} \sum_{m \in M_h} \sum_{a=1}^{A} \frac{w'_a}{\sum_{\hat{a}=1}^{A} w'_{\hat{a}}} g_{a,h}(m)^2 \tag{6}$$

where w'_a is a weighting factor denoting the relative importance of non-continuous changes in variable a. w' is defined at design time.

The discontinuity measure's validity depends on the particular choice of w'. It is well suited to applications such as virtual environments, where users are very sensitive to non-continuous changes in variables. G_h can be computed separately at each host, with the computational complexity being of $O(A \cdot |M_h|)$.

4.3. Yield Measures

Yield measures attempt to measure the influence of inconsistency on the decision-making process of the users of a distributed system. As we've argued before, these influences are highly application-dependent. They are usually difficult to formulate in terms of a mathematical formula. Instead, yield measures are chosen heuristically for each application in an attempt to express the "goal" of an application in a single value.

For example, in the virtual soccer game, one could use the number of bad passes per second as the yield measure. This would be based on the hypothesis that a less effective consistency algorithm would cause players to make more mistakes, e.g. because they've got incorrect information about the potential receiver of a pass. There is usually more than one possible choice of measure, which can yield different rankings for the same set of algorithms.

Yield measures can be applied to any system in which at least a subset of the hosts is accessible for data gathering. Their validity depends on the choice of a particular measure, and there is no general rule for how to come up with a good one. Yield measures are typically chosen so that they are simple to implement. Their computational complexity depends on the specific measure, but typically scales with $O(N \cdot T)$.

5. Experiment setup

After defining several measures, two questions arise. Firstly, can the measures be used to rank algorithms that try to reduce the inconsistency? And secondly, is this ranking consistent with the user impression? In this section, we briefly describe the experiment setup we used to answer these questions. For a full description we refer to our technical reports[2] [4, 11].

Our benchmark application for the evaluation is "Swarm", a non-cooperative game for N players (here: 8). Each player controls a bee and tries to stay as close to the queen-bee as possible. At regular intervals, honey is awarded to each bee depending on its distance to the queen, bees closer to the queen receive more honey. When a bee collides with another bee or with the queen, it is stunned for a few seconds: the bee receives no honey and cannot control its flight. The collisions are modeled as for ideal, fully elastic balls, comparable to a billiard game. One should point out that even small differences in the positions and the velocities of the bees result in a totally different behavior of the Swarm. Each bee is controlled by an artificial intelligence (AI) which keeps it as close to the queen as possible while evading other bees, by changing the acceleration and the direction of the bee. The AI tries to maintain a minimum safety distance S to the queen and other bees to reduce the risk of collisions. S is adapted to the progress of the game in an attempt to optimize the overall honey gathered by this bee, i.e. it is increased if there are many collisions and decreased when there are few collisions.

The yield measure for Swarm is honey per second. If a bee experiences too many collisions due to inconsistency, it is stunned more often, and receives less honey. The bee will react by increasing its safety distance S, which further reduces the amount of honey it collects.

We use the divergence and discontinuity measures from sections 4.1 and 4.2, with all the $r_{a,h}$ set to 1. We have normalized the variable values to their natural domain ranges before using them in this measure. We chose the set of variable weighting factors w_a (cf. eqn. 3) and w'_a analogous to how a user would rate the differences, i.e. more visible or interesting variables are weighted higher.

We've implemented two different consistency algorithms based on the loose and optimistic approach. Loose consistency as implemented in Adam is inspired

[2] available from `http://www.math.tu-berlin.de/condel/publications`

by the algorithms used in DIS and related systems [7]. For a complete algorithmic description, see [4]. Optimistic consistency as implemented in Adam is inspired by some of the newer approaches to distributed game consistency as seen in [10] or [8]. For the exact algorithm, refer to [4] and [11].

The Internet is modeled abstractly by connecting each host directly with each other host. On these links, packets are lost with a probability of 1 ‰. The packet delay is nearly fixed, but 1 % of the packets experience a high delay, for details see [11]. Congestion is not modeled, but we measure network load, i.e. the number of packets sent by each host within a second.

We performed a simple user evaluation test for our sample scenario. Test users would look at the visualizations of the experiments running the Swarm with the two different consistency algorithms, and would decide which one performs better. These results are preliminary, and will be expanded in the near future.

We ran two experiments, each repeated seven times, in our simulator Adam [11] – the Swarm scenario with eight bees for loose and optimistic consistency – for 5000 s with identical settings for loss rate at a fixed network load of 20 packets sent/s by each host. A packet is typically less than 100 bytes in size. We vary the expected network delay between 0 and 500 ms to find out how the different consistency algorithms react to changes in connection quality.

6. Results

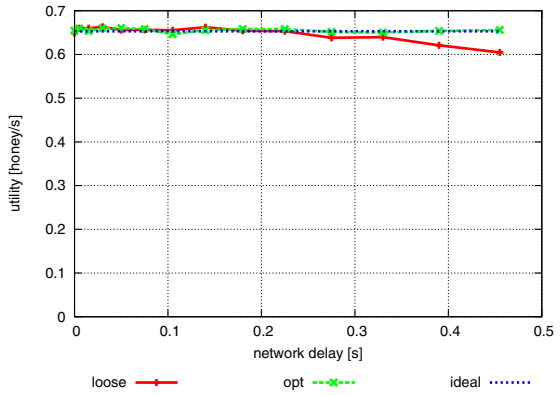


Figure 1: Yield measure comparison – higher is better

User evaluation tests consistently rate optimistic consistency as better than loose for a network delay of 50 ms. Videos showing the first 300 seconds of both visualizations are available from our web server [3], and support this observation.

Figures 1, 2a and 2b show the results of the three inconsistency measures we've proposed. As one can see, both the divergence measure and the discontinuity measure give very similar results which confirm the user evaluation results by consistently rating optimistic consistency better than loose consistency over the whole range of network delay. Optimistic consistency achieves a lower inconsistency, because it incorporates late messages instead of discarding them. The difference grows for higher network delays, which implies that optimistic consistency can cope better with high latency connections. The plot for the yield measure shows that the difference is less obvious than for the other two measures, and becomes manifest only for network delays larger than 350 ms. This is somewhat surprising, as one would intuitively think that inconsistencies in the system should have a strong negative impact on the ability of the bees to collect honey. We suspect that this is due to the large amount of honey awarded for being close to the queen, despite the penalty for more frequent collisions.

Overall, the results confirm our expectations that optimistic consistency always outperforms loose consistency in terms of inconsistency. Furthermore, two of our three proposed inconsistency measures agree with user evaluation results, showing lower inconsistency for the optimistic algorithm as well. Only the yield measure is largely indifferent, showing that inconsistency in the system has a much stronger impact on user perception than can be expressed by the yield measure alone.

The increased consistency provided by the optimistic algorithm comes, however, at the price of increased CPU load which we have found to be 4 times higher for our implementation of optimistic consistency compared to loose consistency.

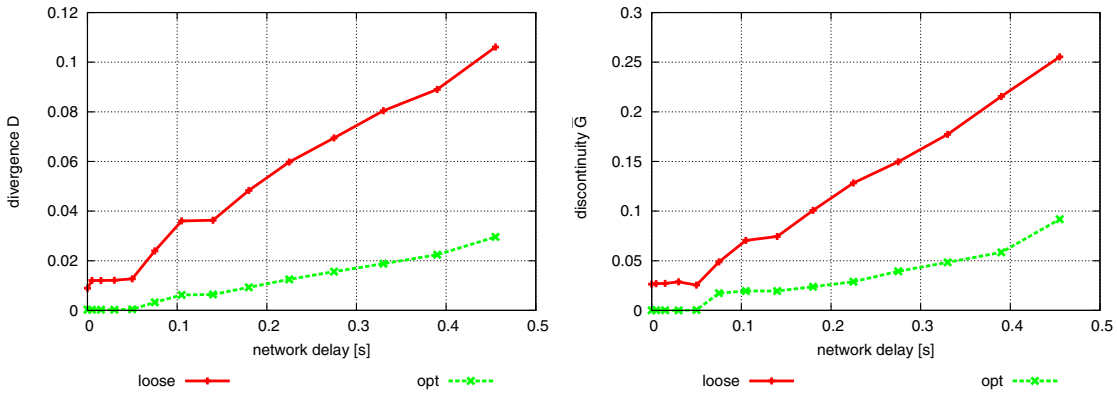
7. Conclusions and Outlook

In this paper, we have discussed the effects of inconsistency on distributed virtual environments and its perception by the user. To quantify these effects, we've proposed three different measures: the divergence measure, the discontinuity measure and the yield measure.

We've defined a minimalist multiplayer game which, while simple, retains the essential properties of a multi-user DVE.

We've applied our measures to two implementations of the game using different consistency algorithms. We've also performed a user evaluation test to compare our experimental results against. We've found that both the divergence measure and the discontinuity measure correlate well with the user evaluation results, while the yield measure fails to do so. We believe that

[3] `http://www.math.tu-berlin.de/condel/visuals/comparison.html`



(a) Divergence Comparison

(b) Discontinuity Comparison

Figure 2: Comparison of measures – lower is better

this shows that yield alone, while intuitively acceptable, is not always sufficient to express the strong effects of inconsistency on user perception.

As a result from our work in this paper, we advocate the use of the Swarm as a benchmark application and the divergence and discontinuity measures as inconsistency measures when comparing different consistency algorithms with each other.

For the future, we plan to apply our measures to a larger set of sample scenarios so that we can evaluate them for a broader range of applications.

We also plan to perform larger and more sophisticated user evaluation tests.

Finally, we will research the possibility of combining the inconsistency measures we've proposed. We believe that this will help to measure effects which cannot be captured by a single measure alone.

References

[1] G. Armitage. An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3. In *Proceedings of the 11th IEEE International Conference on Networks (ICON 2003))*, pages 137–141, Sydney, Australia, September 2003. IEEE Press.

[2] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures (extended version). *Kluwer Multimedia Tools and Applications*, 23(1), May 2004.

[3] R. Galli. *Data Consistency Methods for Collaborative 3D Editing*. PhD thesis, Universitat de les Illes Balears, Nov. 2000.

[4] S. Grottke, J. Sablatnig, A. Köpke, J. Chen, R. Seiler, and A. Wolisz. Consistency in Distributed Systems. TKN Technical Report Series TKN-08-005, Telecommunication Networks Group, Technische Universität Berlin, Feb. 2008.

[5] M. P. Herlihy and J. M. Wing. Axioms for Concurrent Objects. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 13–26, 1987.

[6] D. Liang and P. Boustead. Using local lag and timewarp to improve performance for real life multi-player online fames. In *Proceedings of Netgames'06*. ACM, 2006.

[7] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. In *Proceedings of the 1995 IEEE Virtual Reality Annual Symposium*, pages 2–10, 1995.

[8] 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, Feb. 2004.

[9] C. E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. On Maintaining Interactivity in Event Delivery Synchronization for Mirrored Game Architectures. In *Proceedings of the Global Telecommunications Conference Workshops (IEEE GlobeCom)*, pages 1183–1187, November 2004.

[10] D. J. Roberts and P. M. Sharkey. Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays. In *Proceedings of the IEEE Workshops on Enabling Technology: Infrastructure for Collaborative Enterprise '97*, pages 161–166, 1997.

[11] J. Sablatnig, S. Grottke, A. Köpke, J. Chen, R. Seiler, and A. Wolisz. Adam – A DVE Simulator. TKN Technical Report Series TKN-08-004, Telecommunication Networks Group, Technische Universität Berlin, Feb. 2008.

[12] S. Zhou, W. Cai, S. J. Turner, and H. Zhao. A Consistency Model for Evaluating Distributed Virtual Environments. In *CW '03: Proceedings of the 2003 International Conference on Cyberworlds*, page 85, Washington, DC, USA, 2003. IEEE Computer Society.