Adam – A Testbed for Distributed Virtual Environments

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

Sven Grottke
Institute of IT Services
University of Stuttgart
sven.grottke@iits.uni-stuttgart.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 Enironments (DVEs) the data on which the hosts operate is not consistent at all times. To restore data consistency, the DVE has to employ a consistency algorithm. Unfortunately, all existing DVEs have been built for specific application scenarios, which makes it impossible to compare the consistency algorithms and to choose a suitable candidate for a new scenario.

To overcome this, we have created a modular simulator-based DVE testbed named Adam *with the ability to plug in different application scenarios as well as different consistency algorithms and network constraints. The testbed also contains a large set of measurement tools.*

Our testbed currently supports two application scenarios and several of the most common consistency algorithms found in the literature. We can compare the solutions on an objective scale and confirm that optimistic consistency typically outperforms loose consistency.

1. Introduction

DVEs simulate a virtual world in which users can interact with each other or with that world. An example would be a virtual soccer game where each user controls a player and can interact with other players and the ball. We consider the situation where each user operates a single computer (a host) that renders the world for this user. Through a network, the virtual world is shared between all hosts and the users can observe and react to other users' actions. Immersion into a virtual environment requires very quick local reactions of less than the round-trip delay on many of the relevant networks such as the Internet. Thus, the hosts will typically have to react to local user commands *before* this action is agreed upon by all hosts. This implies that the users on the other hosts act on a different, i.e. *inconsistent* world view. Even if no direct conflict occurs, the inconsistency in world views between the hosts can be detrimental.

A number of consistency algorithms are used to reduce the system inconsistency at the expense of additional network and CPU load. The effectiveness of the consistency algorithms depends both on the properties of the application scenario as well as on network constraints and available hardware.

Even though the usability and overall quality of a DVE depend heavily on its inconsistency, there is currently no ranking of effectiveness of the consistency algorithms used in DVEs. Therefore, it is very hard to estimate the expected inconsistency, network and CPU load of these algorithms for a new application scenario.

In this article we propose a testbed for objective ranking of consistency algorithms for given application scenarios. This code implements some of the most common consistency algorithms used in DVEs and a few simple scenarios. The testbed is completely modular, allowing any combination of scenario, consistency algorithm, and network constraints.

The code also includes a large set of measurement tools. In addition to network and CPU load, several measures are provided to quantify the system consistency, see section 4 for more on these.

1545-0678/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.Workshops.2008.65

The code will be downloadable and extensible so that other researchers can rank their own algorithms.

2. Related Work

The first major DVE was SimNet/NPSNET, which became the DIS standard in 1993[13]. A large array of research DVEs were built similar to DIS, with some extensions. These extensions usually focused on exploiting locality, such as SPLINE [1] or DIVE [6]. DIS was originally designed for military battlefield simulations on dedicated networks and very few of the DIS-alikes are in production use outside of this application.

On the other hand, computer games such as "X-Wing vs. TIE Fighter" (1997) [12] gauged Internet connected simulations early on. "Ultima Online" (1997) connected thousands of players on a single server, "World of Warcraft" (2004) had 9 million subscribers in 2007. Surprisingly, the consistency algorithms used have not changed much between the latter two programs and are akin to DIS, except that centralized servers are used.

The research game application PaRADE (1997) was highly interactive and attempted to solve the Internet-delay problem through optimistic consistency and through prediction [16]. MiMaze (1999) was another very interactive research game application, but it used DIS-like mechanisms [4]. More recently in 2002, Cronin et al. researched algorithms to improve the consistency of an existing, highly interactive game, using optimistic consistency to achieve this [3].

Since the current research focus is most often on scalability, i.e. the ability of a DVE to support many thousands of players at the same time, researchers have started to abandon user tests as too expensive, running simulations instead. RING already simulated 1000 users in 1995[7]. While Mercury (2002) simulated only 100 hosts[2], Knutsson (2004) simulated 4000 hosts[9]. However, all of these simulations were used to find the effect of a specific algorithmic change or of an algorithmic parameter in an otherwise monolithic application and are completely nontransferable to other scenarios or other algorithms.

3. The Testbed

Adam, our testbed, simulates an entire DVE on a single computer. This includes the network connecting the hosts, all the hosts, and even the users. On each simulated host, consistency algorithms are run as they are on a host in an actual DVE, updating the world view of that host. The user operating that host is simulated by running an intelligent agent (*IA*) which bases its decisions on the current world view of the host. Rendering, user interface, system compatibility, patching, etc. are not simulated on the hosts, as there are no actual users or actual hosts.

3.1. Architecture

Adam is built on top of the discrete event simulator OMNet++ [17] in conjunction with the MRIP engine Akaroa [5].

During the simulation, our testbed creates a host-module for each host to be simulated and connects them via OMNet++ network connections, allowing OMNet++ to simulate the entire connecting network. Each host-module then creates a number of modules, the most important of which are shown in figure 1:

Host: A representation of a single physical host the DVE runs on. It instantiates and controls the other modules and is also the OMNet++ interface.

Router: The network layer on a host. Implements reliable transfer algorithms as well as message aggregation if needed.

Consistency: The consistency layer on a host. This decides when to send update information to other hosts and what to do with incoming information. Often, this will hold additional, internal world views.

World: The world view on this host. All objects in this world and their states are collected here as a set of named variables.

Scenario: The scenario-specific code. This contains the rules dictating how objects in this world move about or interact. It also contains a scenario-specific intelligent agent that issues user-commands.

All modules are called at regular intervals to update themselves and clean up any outstanding operations. Amongst other things, this allows the scenario to update the world view according to the scenario rules, thus progressing the virtual environment. Also, the IA decides on new actions during these calls.

When the scenario needs to change the values of any variable in the world view, it can simply do the change directly. If the scenario's IA, however, performs a user event, it will ask the consistency algorithm to perform the event instead. The consistency algorithm can then apply the event directly or delegate it for later application. The consistency may create a message to inform other hosts of this event. If so, the message is passed to the router which then decides when and how

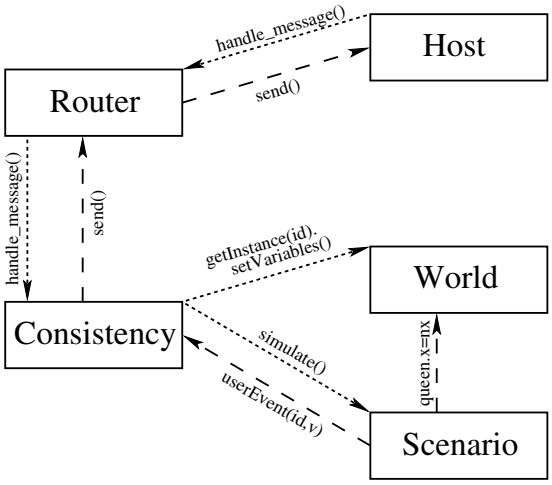

Figure 1. Modules in a Single Simulated Host

to send this message to other hosts via the host's OMNet++ interface.

When a message arrives at a host, it is passed to the consistency layer. The consistency layer analyzes the type of message. Depending on the algorithm in use, it may then drop the message, or add it to its message history, or implement the effect of the message directly by applying it to the world view. Afterwards, it may also call the scenario to update the world view.

The modules described above are implemented as C++ classes. The actual implementations (e.g. a Swarm scenario, or a loose consistency) are then inherited from these base classes, overloading their major interface functions. By using OMNet++ configuration files the user then decides which combination of objects to combine and sets their parameters. This allows easy access to any mix of available technologies, algorithms and scenarios.

One of the few things implicitly shared between the hosts is an exact notion of time (i.e. all hosts use OMNET++'s simTime()). This is not possible in the real world, and in fact it is somewhat difficult to keep a large set of hosts close to the wall-clock time. It is, however, *possible* to do so, at least with respect to message causality[11]. The NTP protocol accomplishes a synchronization of within 10ms[15]. We therefore decided to abstract from this problem.

3.2. Network

We use a flat network model where every host can communicate with any other directly. We set the message delays on these links to be random i.i.d. While this precludes congestion modeling, we *do* measure network traffic on each host, so it is possible to catch possible congestion situations a posteriori. As long as the traffic generated by the considered DVE would not be a substantial part of the overall traffic on the Internet, this abstraction is sane.

The delay probability distribution of the network links was modeled according to our measurements of actual Internet pings[10]: Most of the messages arrive after a fixed delay, the rest coming in within the next 2s.

In addition to the delay specified above, there is a configurable probability a packet may be lost altogether. On the modern Internet, this chance appears to be rather small, no more than 1 ‰[10].

3.3. Consistency Algorithms

Our testbed currently implements three consistency algorithms:

3.3.1. Ideal. For upper bound comparison purposes, we provide an ideal consistency, in which all variables are kept consistent on all hosts.

3.3.2. Loose. Loose consistency as implemented in Adam is inspired by the algorithms used in DIS and related systems [13], including player/ghost analysis and dead reckoning. For a complete algorithmic description, see [8].

3.3.3. Optimistic. Optimistic consistency as implemented in Adam is inspired by some of the newer approaches to distributed game consistency as seen in [14], including reliable communication protocols, local lag and a rollback mechanism based on the so-called trailing state synchronization as described by [3]. For the exact algorithm, refer to [8].

3.4. Scenarios

As of now, the following basic scenarios are provided by Adam:

3.4.1. Pong. The first scenario we implemented was Pong, akin to the famous video-game from 1972. While this works as a model for ball games, it features only two players.

3.4.2. Swarm. The Swarm models a non-cooperative game for N players, each controlling a bee and trying to stay as close to the queen-bee as possible. At regular intervals, honey is awarded to each bee depending on their distance to the queen, with higher distance getting less of a reward. When a bee collides with another or with the queen, the bee is penalized by being considered

dead for a few seconds. While dead, a bee receives no honey and cannot control its flight.

The game employs a physical model. Players cannot simply set position or speed of their bees, only the direction and amplitude of acceleration. Collisions between bees/queen/walls are resolved as they would for ideal, fully elastic balls. In particular, a crash often leads to more crashes as the bees bounce on uncontrollably.

When passive replication is used (for loose consistency), each bee's score is owned by the next bee for fairness reasons. The queen's ownership is passed round-robin every two seconds.

The IA's algorithm a strong influence on the measurements in this scenario, therefore we made an effort to improve the IA enough so that it would behave similar to a human. A simple version of potential-field steering is installed which attempts to solve the tradeoff between trying to move near the queen while trying to keep a safe distance from other bees and the queen, so as not to collide with anything. A human player would quickly adapt to the game and simulation quality and adjust his safety margins accordingly. Therefore, the IA was also outfitted with a simple learning mechanism to tune its safety margins.

4. Inconsistency Measures

To compare different consistency algorithms with each other, Adam provides several measures for the degree of inconsistency in the virtual world:

4.1. Divergence

This measure estimates how similar the world views are on the different hosts. It is based on the standard deviation of the values of each variable v across the hosts. For each variable a, we define the divergence measure at time t as

$$d_a(t) := \sqrt{\frac{1}{H}\sum_h \left(v_{a,h}(t) - \overline{v_a}(t)\right)^2}$$

where $v_{a,h}(t)$ is the value of variable a on host h and H is the number of hosts. $\overline{v_a}(t) := \frac{1}{H}\sum_h v_{a,h}(t)$ is the average value of variable a across all hosts.

These per-variable snapshot divergences are then weighted and averaged into a single overall system divergence

$$D = \frac{1}{T}\sum_t \sum_a \frac{w_a}{\sum_a w_a} d_a(t)$$

, where w_a are configurable weighting factors that allow us to emphasize the more important variables (such as spatial differences, readily visible) versus the less important variables (such as acceleration differences, which are almost invisible to the naked eye). T is the amount of snapshots taken, this makes the measure independent of the overall time the system was measured.

The value given by D is an objective judgment of how well the world views of the different hosts coincided during the DVE's lifetime. A smaller D indicates better consistency.

4.2. Discontinuity

This measure estimates how much the DVE changes discontinuously. The motivation is that when a host receives a network message from another host, the host has to change his own world view. If the current world view is visible to the user, this change is also visible. Most humans are very sensible to such effects and find them very unpleasant.

Whenever a network message causes a change in variable a on host h's world view, the change is expressed by

$$g_{a,h}(m) := |v^{\text{new}}_{a,h}(m) - v^{\text{old}}_{a,h}(m)|$$

, where m indexes the discontinuous changes. The total discontinuity is then weighted and averaged across the hosts:

$$\overline{G} := \frac{1}{H}\frac{1}{T}\sum_h \sum_{m_h} \sum_a \frac{w'_a}{\sum_a w'_a} \cdot (g_{a,h}(m_h))^2$$

, where m_h indexes all changes made for host h and T is the total time of the experiment. w'_a are weighting factors similar to the ones in chapter 4.1.

$\overline{G}$ describes the average discontinuity on the entire system. Smaller values indicate a smoother play out.

5. Experiments

We are running Adam on standard desktop PCs (3 MHz Intel CPU, 1 GB memory). The computational complexity of the Swarm scenario is $O(N^3)$, where N is the number of hosts. For practical purposes, this limits the number of hosts that can be simulated to sixteen. The theoretical complexity of the simulation is $O(N^2)$, but Adam has not been optimized to this respect yet.

We ran each Swarm experiment with eight simulated hosts, averaging the results between seven runs of each 5000s.

5.1. Loose vs. Optimistic

We measured parametric curves for our inconsistency measures over network load. To create these

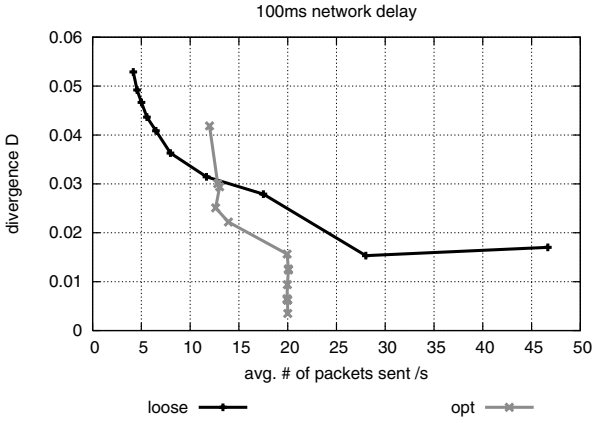

Figure 2. Divergence Characteristic for 100ms Network Delay

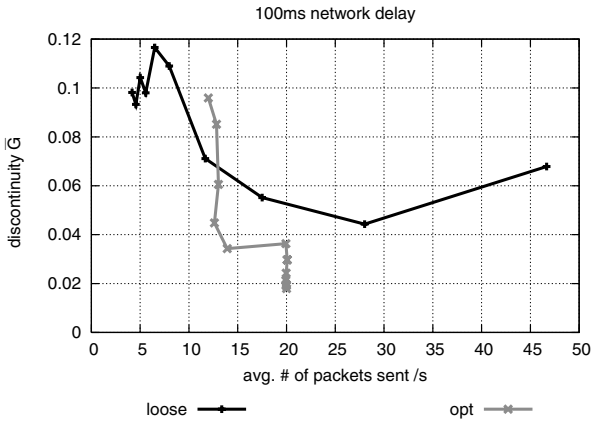

Figure 3. Discontinuity Characteristic for 100ms Network Delay

curves, we stepped through a single parameter in the Swarm experiment, leaving the other parameters at their defaults. For loose consistency, we stepped through regular update rate, while for the optimistic case, we stepped through the message aggregation timeout. The resulting network load and inconsistency measurements from each sub-experiment are then plotted in figures 2 and 3 for a network delay of 100ms and in figures 4 and 5 for the 400ms case.

The resulting characteristic curves not only allow an estimate of the tradeoff between inconsistency and network load, they also allow a direct comparison of the two consistency-algorithms used, loose and optimistic.

From the graphs, it is immediately apparent that optimistic consistency results in significantly lower inconsistency for the entire range of bandwidth > 13 packets/s, in all sub-experiments. We will analyze this effect further to find exact criteria where optimistic consistency is superior to loose consistency.

Optimistic consistency cannot be configured to use *less* than a certain critical bandwidth (13 packets/s in this scenario), so if the desired network load lies below that, loose consistency must be used.

Optimistic consistency also cannot be configured to use *more* than a certain bandwidth. However, the consistency at that point is already significantly better than loose's consistency at any network load.

Loose consistency, on the other hand, can run at any desired bandwidth. At extremely low network load, its inconsistency behaves roughly as 1/x. At high network load however, it fails to profit from any additional network load and its inconsistency becomes constant.

Note that this result, i.e. that loose consistency is only better than optimistic consistency at extremely low bandwidth, has been observed before[14], but this is the

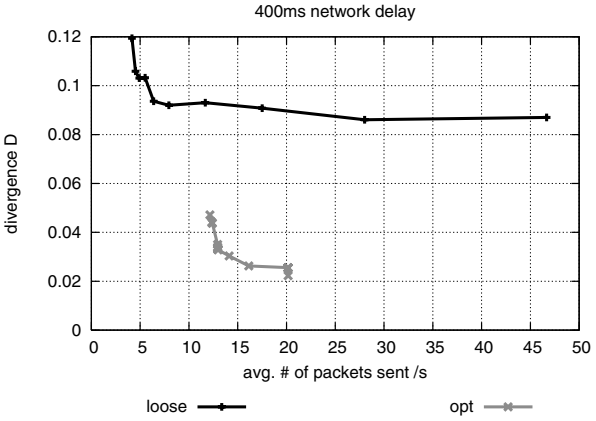

Figure 4. Divergence Characteristic for 400ms Network Delay

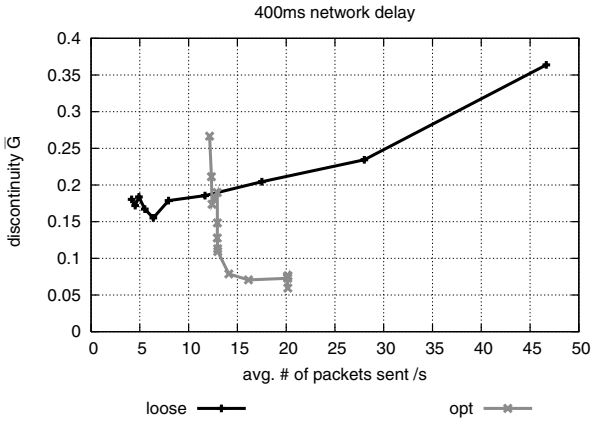

Figure 5. Discontinuity Characteristic for 400ms Network Delay

first time that a quantification of the effect is available.

6. Conclusions and Outlook

We have shown that our testbed can be used to directly compare specific consistency algorithms with each other. We implemented various consistency schemes, scenarios, measures and standard experiments.

We used these experiments to confirm that for a DVE using a physical model, optimistic consistency typically performs better than loose consistency. This is the expected result and thus validates Adam's measurement capabilities. We are planning to perform user evaluation tests to confirm that Adam's measurements do indeed correspond to perceived quality.

The computational complexity of Adam's current Swarm scenario algorithm limits simulation to sixteen hosts. In the future, we plan to upgrade Adam to up to a hundred hosts while maintaining the current consistency algorithms.

A major challenge in research on virtual environments is currently the question of how to create scalable solutions supporting thousands of hosts. Classical DVEs scale with the number of hosts both in computational complexity per host and network load per host. Thus, a scalable DVEs must employ partial replication, i.e. allow each host in the DVE to compute only a part of the world and communicate with only a part of the other hosts.

We are planning to install the most important existing replication algorithms, such as classic multicast and current peer-to-peer systems. They will still be implemented in a modular way as everything else in Adam in order to mix each scalability support with each scenario and consistency algorithm. Because of the lower complexity per host in a scalable DVE, we expect the improved Adam to be able to simulate thousands of hosts for such a DVE.

References

[1] J. Barrus, R. Waters, and D. Anderson. Locales and beacons: Precise and efficient support for large multiuser virtual environments. In *Proceedings of VRAIS'96*, pages 204–213, 1996.

[2] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *Proceedings of the first workshop on Network and system support for games*, pages 3–9, 2002.

[3] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin. An Efficient Synchronization Mechanism for Mirrored Game Architectures. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 67–73, 2002.

[4] C. Diot and L. Gautier. A Distributed Architecture for Multiplayer Interactive Applications on the Internet. *IEEE Network magazine*, 13(4):6–15, July/August 1999.

[5] G. C. Ewing, K. Pawlikowski, and D. McNickle. Akaroa2: Exploiting Network Computing by Distributing Stochastic Simulation. In *Proc. European Simulation Multiconference ESM'99*, pages 175–181. International Society for Computer Simulation, June 1999.

[6] E. Frécon and M. Stenius. DIVE: A Scalable Network Architecture for Distributed Virtual Environments. *Distributed Systems Engineering Journal*, 5(3):91–100, Sept. 1998.

[7] T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Symposium on Interactive 3D Graphics*, pages 85–92, 209, Apr. 1995.

[8] S. Grottke, J. Sablatnig, A. Köpke, J. Chen, R. Seiler, and A. Wolisz. Consistency in Distributed Systems. Technical Report TKN-08-005, TU-Berlin, TKN Technical Report Series, February 2008.

[9] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 1, 2004.

[10] A. Köpke and A. Wolisz. Delay Measurements on the Internet. Technical report, TU-Berlin, TKN Technical Report Series, 2008.

[11] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[12] P. Lincroft. The Internet Sucks: Or, What I Learned Coding X-Wing vs. TIE Fighter. In *Proceedings of the Game Developer's Conference*, September 1999.

[13] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. *Presence*, 3(4):265–287, 1994.

[14] 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.

[15] D. L. Mills. Network Time Protocol Version 4 Reference and Implementation Guide. Technical Report 06-06-1, University of Delaware, Electrical and Computer Engineering, June 2006.

[16] 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.

[17] A. Varga. The OMNet++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference (ESM 2001)*, June 2001.