Message Caching for Local and Global Resource Optimization in Shared Virtual Environments

Helmuth Trefftz
Rutgers University
96 Frelinghuysen Road
Piscataway, NJ 08854
USA
(1)(732) 445 0542
Trefftz@caip.rutgers.edu

Ivan Marsic
Rutgers University
96 Frelinghuysen Road
Piscataway, NJ 08854
USA
(1)(732) 445 0542
Marsic@caip.rutgers.edu

ABSTRACT
The use of Shared Virtual Environments is growing in areas such as multi-player video games, military and industrial training, and collaborative design and engineering. At the same rate, different mixes of computing power and graphics capabilities of the participating computers arises naturally as the variety of people/organizations sharing a virtual environment grows. This paper presents an adaptive mechanism to reduce bandwidth usage and to optimize the use of computing resources of heterogeneous computers mixes participating in a shared virtual environment. It is based on caching of both outgoing- and incoming-messages. We also report the results of implementing the proposed scheme in a simple shared virtual environment.

Keywords
Virtual Reality, Shared Virtual Environments, Message Caching, Networking.

1. INTRODUCTION
Shared Virtual Environments (SVEs) allow multiple geographically separated users, to interact in almost real-time [1]. SVEs have been successfully deployed for several years in applications such as military [2,7] and aerospace training [5]. The availability of higher bandwidth, as well as the increasing power and low cost of personal computers with 3D graphic accelerators make new SVE applications accessible for the general population. The applications include education [8, 14], multi-user games [4] and concurrent engineering [9].

As SVEs leave closed and controlled environments, such us military simulations, their design needs to take into account:

- The possibly large number of users that will coexist in the SVE at a given time
- The possibly great heterogeneity of the machines that will run the SVE at a given time.

As the number of users in the SVE increase, so does the bandwidth required to update the position and orientation of moving entities, as well as other types of events that need to be transmitted to maintain a consistent shared state. If the number of participants is too large or the bandwidth too small, the network might easily become a bottleneck [10]. Since the network is a shared resource, the scope of its optimization is global and should be performed at execution time, since the number of users sharing the virtual world may vary in time.

On the other hand, the SVE program should locally adapt to the computing and graphics capabilities of the host machine. There are several parameters that can be fine-tuned in order to maintain an acceptable degree of fidelity while accommodating the limitations of the local machine, such as:

- using less-detailed representations of virtual objects that are outside of the user's area of interest
- reducing the polling frequency of local devices (such as wands, space mice, trackers,etc.)
- reducing the accuracy and/or scope of collision detection
- reducing the frequency of local update of remote events

Generally, the designer of the virtual environment takes into account the type of machines the SVE will run on and creates a program that will run reasonably well on the target computers. The designer is thereby making a static decision with global impact. We argue that the scope of these optimizations should be local and, furthermore, done adaptively at run-time. For example, Michael Capps has recently explored tuning fidelity parameters in Shared Virtual Environments according to user preferences [1].

In this paper we report the results of our experiments on message caching. In a first set of experiments, the parameters that define the caching were set statically in order to isolate the effects on the participating nodes. In the next experiments the same parameters were computed adaptively, changing in time as the simulation advanced.

1.1 Message Caching
One of the most important challenges in implementing an SVE is maintaining a consistent description of the virtual world across the collaborators in the presence of frequent updates. In this sense, the differentiating factor among the architectures is the location where the database describing the current state of the virtual world is maintained. At one end of the spectrum are the **centralized** systems, in which the description of the virtual world is kept on a server. The server broadcasts the virtual world updates to the clients. On the other end of the spectrum are the **replicated** or **peer-to-peer** architectures, in which the

status of the virtual world is reproduced in every participating site. Each node is responsible for keeping the local copy consistent and up-to-date. We have taken the latter approach for the system described in this paper. There are numerous combinations in between these two extremes, such as systems that assign a "home node" to each entity. In this case, the state of the entity is maintained in the home node only.

The challenge of maintaining a consistent description of the virtual world is similar to many problems in distributed systems, such as the problem of consistency in distributed databases or the cache-coherence problem in multi-processor machines; and designers of SVEs can apply similar solutions.

Aggregation of messages is described in [12] as a technique to reduce bandwidth utilization by grouping together sets of packets sent to the network. Bandwidth saving takes place since only one header is used for the whole group. Singhal reports that the aggregation can be done according to time (sending the update whenever a pre-defined timeout expires) or to packet number (sending the update whenever a certain number of packets be ready).

The **message caching** scheme presented here is different in several ways as will be detailed later. The general architecture of our system is described in Figure 1. The program was written mostly in Java and Java3D [13], except for the driver of the Space Mouse, which was written in C. The main components of the system are:

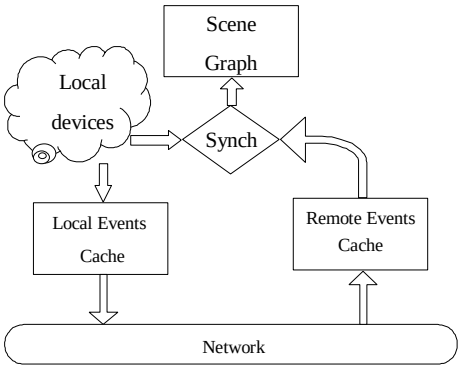


Figure 1: Architecture of the system

- **Scene Graph**: The data structure that contains the current state of the shared world. The display thread (Java3D) reads this data structure and renders the scene from the user's virtual viewpoint.
- **Local Devices:** Drivers for the local devices used for interaction in the virtual world.
- **Local Events Cache (LEC):** Cache for locally generated events. An entry is kept in the outgoing-messages-cache for every entity that has been modified locally. Local updates are displayed and reflected in this cache as soon as they occur, but they are not sent to the network immediately. Instead, an outgoing-time-out (OT) is defined and the updates are sent when the time-out expires. In the meantime, any new update to a local entity overwrites any older updates to the same entity existing in the Local Events Cache.
- **Remote Events Cache (REC)**: Cache for remotely generated events. An entry is kept for every remotely modified entity. Again, remote updates are not displayed immediately. Instead, an incoming-time-out (IT) is defined and the updates are reflected in the local scene when the time-out expires. In the meantime, a new update to a remote entity overwrites any older updates to the same entity existing in the IMC.
- **Synch**: Object that guarantees mutually exclusive access to the Scene Graph

Our approach is different from the one described in [12] in the following essential ways:

- Our scheme handles not only out-going messages but also in-coming ones.
- We propose not only grouping the messages but also sending only the latest update for each modified entity. Thus, our scheme behaves more like a cache.

The global and local impacts of the OMC and the IMC are explained next.

1.1.1 Caching of Outgoing Messages

Caching the local updates before sending them to the network has a *global* impact on the use of the network. Most of the time, users in an SVE are either moving in the world or manipulating an object. In both cases, the updates will be cached in the LEC and, depending on the value of the OT, the number of updates sent to the network will be reduced considerably. A larger value of OT will avoid having a fast machine flooding the network and the slower machines with too frequent updates.

The frame rate (number of times the scene is rendered in the local computer per unit of time) is important when determining the value of OT and IT. The frame rate of the fastest machine defines a lower boundary for OT. Sending updates more frequently than the fastest machine can display them would be a waste of resources. Defining OT as the reciprocal of the frame rate of the *slowest* machine would make the updates unnecessarily slow for everybody else. For this reason we decide to define OT as the reciprocal of the median frame rate among the participating machines.

$$OT = \text{median } \{(1/FR_I), I: \text{for all participating nodes}\}$$

where FR_I is the Frame Rate of node I.

1.1.2 Caching of Incoming Messages

Caching remote updates before updating the local scene has a *local* impact. The local data structure describing the Virtual World becomes a critical shared resource that can be updated by either local or remote events in order to avoid inconsistencies.

Caching remote updates favors local updates, in which the user is probably more interested.

Slower nodes use caching of incoming messages to reduce further the flooding effect of frequent updates generated by faster machines. Messages are received and stored in the local cache, but the scene is not necessarily updated immediately. Participating computers with a faster frame rate than the reciprocal of OT do not need to slow down the rate of remote updates, but slower computers do, in order to favor local updates. For this reason we are using the following formula for IT:

$$IT = \max \{(1 / FR) * (N + 1), OT\}$$

where FR is the Frame Rate of the node and N is a factor related to the importance of local events relative to remote events.

In a system that polls the devices once per frame, if N is 1, IT will tend to accommodate for 1 local event for every remote event; if N is 2, IT will accommodate for 2 local events for every

3. Results

We observed a positive impact of our scheme in the following areas:

- Reduced bandwidth requirement.
- Increased frequency of local updates
- Increased frame rate.

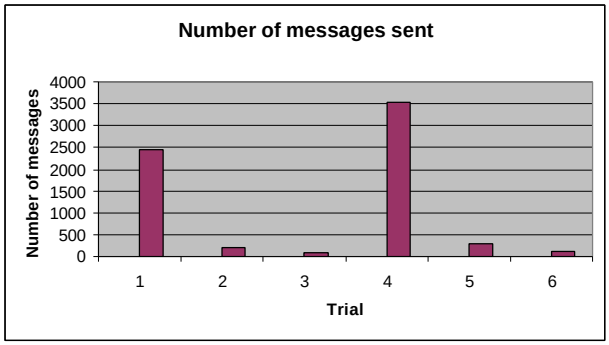
The impact is explained in the following sections.

3.1 Static Behavior

In the initial set of experiments, values for IT and OT were assigned statically as parameters when starting the application.

3.1.1 Reduced Bandwidth

As mentioned before, the effect of caching out-going messages is reducing the amount of utilized bandwidth. This rather evident fact is shown in Graph 1, which describes the number of messages sent by all participating computers during the trials. The number of messages is larger for trials 4, 5 and 6 since three computers are running, instead of two.

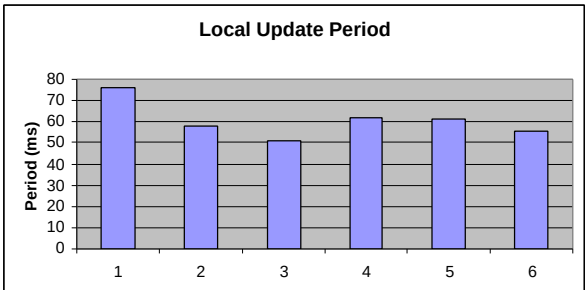


Graph 1. Messages sent per trial

3.1.2 Local Updates

In the presence of caching, the user actions are reflected faster on the local scene graph, as described in Graph 2. Locally generated updates are given priority over remotely generated events as the caching time-out times increase.

Note that the local Update Period is considerably smaller than the Frame Period for Dragonfire, the slowest computer in the setup. In our system, local devices are handled by a separate thread, instead of being polled once per frame. This allows for more frequent generation of events. From a local perspective, this is a waste of resources, but it allows faster computers sharing the same environment to receive more frequent updates.



Graph 2. Local Update Period in milliseconds for Dragonfire.

3.1.3 Frame Rate

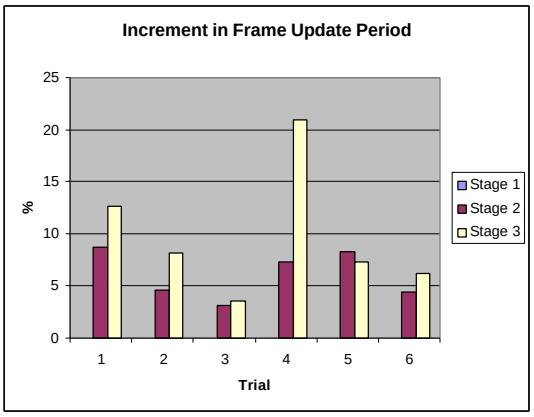
The frame rate of slower machines is affected adversely as the number of remote events increases. Caching reduces the negative impact, bringing the frame rate closer to the baseline level, as described in Table 3 and Graph 3. Table 3 shows the average frame rate period for **Dragonfire** during each stage and Graph 3 shows the percentage increment of stages 2 (only local updates) and 3 (local and remote updates), relative to stage 1 (baseline).

In trials 1, 2 and 3, two nodes were participating. in trials 4, 5 and 6 three nodes were participating. It is easy to see the trend towards larger frame rate periods as the number of nodes increase. The most dramatic increment is 20%, when three nodes take part of the environment without message caching.

On the other hand, the impact of the remote events is reduced as the caching time-out values grow from 0 (tasks 1 and 4) to 250 ms. (task 2 and 5) and to 500 ms. (task 3 and 6).

Table 3. Average Frame Rate Period in milliseconds

Trial	Stage 1	Stage 2	Stage 3
1	1042	1133	1174
2	1197	1252	1294
3	1125	1160	1165
4	1170	1255	1415
5	953	1032	1023



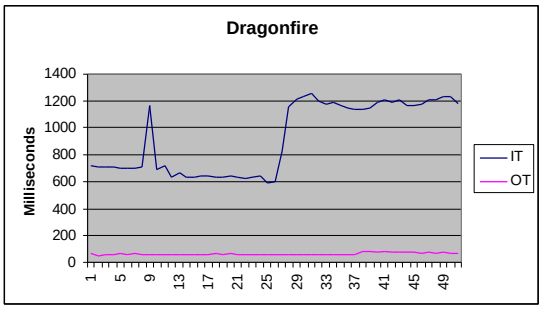
Graph 3. Percentage increment in frame period

3.2 Adaptive Behavior

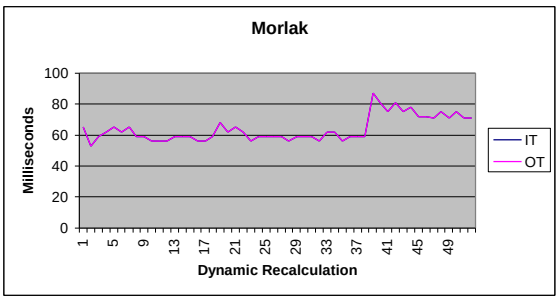
In the next set of experiments, the values for IT and OT were calculated at run time, according to the formulas defined in section 1.1.1 and 1.1.2.

In our implementation a separate thread is used to read the input devices, and on slow machines, such as Dragonfire, the threads can read the device considerably faster than the frame rate. Variable N in the computation of IT was assigned 0 in order to avoid slowing down unnecessarily the display of local updates.

Graphs 4 and 5 show the adaptive values of IT and OT for Dragonfire and for Morlak. The value of OT for all participating nodes was basically determined by Morlak's frame rate: it was always the median value since Bochica was faster and Dragonfire slower. Note that curves for IT and OT overlap for Morlak, since both are determined by its frame rate.



Graph 4. Values of IT and OT for Dragonfire



Graph 5. Values of IT and OT for Morlak

Bochica's IT value was also determined by Morlak's frame rate. Even though Bochica could display updates faster than Morlak could produce them, it would make no sense trying to update messages at a faster rate than any node produced them.

Dragonfire's IT, on the other hand, was only determined by its own frame rate, which was considerably slower than OT. It would make no sense updating the scene data structure faster than the rendering thread could display it. The large change in Dragonfire's frame rate around update number 29 corresponds to insertion and manipulation of objects by remote machines.

One interesting result of the combinations of IT and OT values was that updates originated in Dragonfire were displayed more frequently in the remote machines, which had faster graphics hardware, than in Dragonfire itself. This demonstrates the utility of decoupling frame rate and device polling.

4. Conclusions and Future Work

We have implemented an adaptive message-caching scheme that can be used, at run time, to perform global and local optimizations of resource utilization. Our initial experiments show the positive impact on several critical factors of SVEs, most interestingly bandwidth requirements and display frame rate. Our results also show how to adaptively adjust the parameters that control the caching in order to accommodate for varying conditions during the simulation.

Our main interest is to enable effective collaboration in Shared Virtual Environments even when the participating computers have varying degrees of computing power. We plan to devise similar schemes to work with other parameters such as level-of-detail and frequency to poll local devices, always with the objective of adaptively optimizing both local and global usage of resources at run-time.

5. REFERENCES

[1] M. Capps. Fidelity Optimization in Distributed Virtual Environments. Ph.D. thesis. Naval Postgraduate School, Monterey, California. June 2000.

[2] D. Cohen. DIS-Back to basics. In *Proceedings of the 11th Workshop on Standards for Distributed Interactive Simulations*, 603 – 617. September 1994.

[3] K. Hartung, S. Muench, L. Schomaker, T. Guiard-Marigny, B. Le Goff, R. MacLaverty, J. Nijtmans, A. Camurri, I. Defee and C. Benoit. "Development of a System Architecture for the Acquisition, Integration and Representation of Multimodal Information", Deliverable 4, ESPRIT Project 8579 – MIAMI, 1996.

[4] idSoftware Web site: http://www.idsoftware.com

[5] R.B. Loftin and P. Kenney. The Use of Virtual Environments for Training the Hubble Space Telescope Flight Team. In http://metropolis.vetl.uh.edu/Hubble/virtel.html

[6] Magellan/Space Mouse User's Manual. LogiCad3D GmbH, a Logitech company. Gilching, Germany, 1999.

[7] NPSNET Web Site: http://www.npsnet.nps.navy.mil/npsnet

[8] V. Pantelidis and L. Auld. VR in the Schools. In http://www.soe.ecu.edu/vr/pub.htm

[9] M. Roussos, A.Janson, J. Leigh, C. Barnes, C. Vasilakis, and T. Moher, ``The NICE project: Narrative, Immersive, Constructionist/Collaborative Environments for Learning in Virtual Reality," In *Proceedings of ED-MEDIA/ED-TELECOM 97*, Calgary, Canada, June 1997, pp. 917-922.

[10] S. Singhal. Using Projection Aggregations to Support Scalability in Distributed Simulation. In *Proceedings of the 1996 International Conference on Distributed Computing Systems*. Hong Kong. 1996.

[11] S. Singhal and M. Zyda. *Networked Virtual Environments*. Addison-Wesley, New York, 1999.

[12] ibid. pp 191 – 195.

[13] H. Sowizral, K. Rushforth, M. Deering. *The Java 3D API Specification.* Addison-Wesley, 2nd Ed, 2000.

[14] H. Trefftz, C. Correa, M. A. Gonzalez, J. Restrepo, C. Trefftz. "Virtual Reality and Distance Learning in Colombia", *IASTED International Conference COMPUTER GRAPHICS AND IMAGING*, Halifax, Nova Scotia - Canada, June 1 - 3, 1998.