A Consistency Model for Distributed Virtual Reality Systems

Vasily Y. Kharitonov
Department of Computers, Systems and Networks
Moscow power engineering institute (technical university), Russian Federation
E-mail: KharitonovVY@mail.ru

Abstract

As a further development of works published in previous DepCoS-RELCOMEX conferences, this paper is devoted to one of the most important problems taking place in distributed virtual reality systems (DVR systems): maintenance of consistency among the processes within DVR system, which is required to ensure that all users have a sense of presence in a common shared virtual world. In this paper an event-based model of distributed computations in DVR systems is discussed. Then we consider different approaches to consistency definition, such as causal, observational and space-time consistency.

In order to reach high space-time consistency in DVR system, large amounts of geographically distributed data should be replicated in real time. However, due to hardware limitations it is still impossible to achieve absolute consistency. Moreover, in most cases the absolute consistency is not necessary. In this work we propose a data consistency model based on selective consistency principle allowing to ensure consistency only on those parameters which are in most importance for specific task. Within the model the issues of data representation, storage and communication are discussed.

1. Introduction

The creation of Distributed Virtual Reality systems (shortly, DVR systems) is one of the most perspective and promising areas of modern computer science. The increased interest in such systems primarily associated with qualitatively new opportunities that they provide to humans. First of all, the user is given the opportunity to «immerse» into a three-dimensional highly-realistic environment (also called a *virtual environment, VE*), which accurately simulates the work environment in which he used to work in the real world. In addition, the user can interact with the objects of the virtual world both visually and through various input/output devices and manipulators. The most important property of DVR systems is the possibility of interaction of many geographically remote users in a common VE. This property enables users to work together, despite how far they are from each other, which can be claimed in many applications of human activity.

One of the major challenges in designing DVR systems is to create the virtual environment which is the best possible approximation to the real world. In DVR systems this requirement means not only creating realistic three-dimensional graphics, but also ensuring the interaction of users in a VE which should be the most accurate representation of users interaction in the real world when dealing with similar problems. In other words, we are talking about the quality of users interaction. But then the question arises: what is meant by "the quality of interaction" and how it could be evaluated.

In this work it is proposed to evaluate the quality of users interaction on the basis of the VE state consistency between different users. First, event-based model of distributed computations in DVR systems is considered. Then, the various approaches to defining consistency in such systems are discussed. And, finally, a data consistency model based on selective consistency principle is proposed.

978-0-7695-3674-3/09 $25.00 © 2009 IEEE
DOI 10.1109/DepCoS-RELCOMEX.2009.17

2. Event-based model of distributed computations in DVR systems

2.1. DVR systems features

DVR systems are a subclass of distributed systems and, therefore, inherit all of their properties [1]:

- *No global physical clock.* All system processes interact asynchronously (but the system can still maintain some clock synchronization model).
- *No shared memory.* All communications are carried out only through the message exchange between processes (though there may be a shared memory abstraction).
- *Geographical separation*. Processing nodes of DVR system can be located within the local area network (LAN), as well as on a wide-area network (WAN).
- *Autonomy and heterogeneity*. Computational nodes are loosely coupled in the sense that they may have different performance and running various operating systems. They are usually not part of a dedicated system, but interact with each other to solve common problems.

However, DVR systems have their own features:

- *All computations are performed in real time.* This imposes special requirements on computational nodes which should be able to handle large amounts of information in short time, as well as on communication channels which should ensure instant access to the most actual state of the VE;
- *Specific nature of distributed computations in DVR systems.* Most of the computations in DVR systems are concentrated on the VE global state calculation which is constantly changing, not only in response to users' actions, but also with the progress of time.
- *Requirement of clock synchronization.* Although there is no common system clock, each process has its own local clock and in DVR system it is very important that these local clocks are well synchronized with each other.

2.2. A Formal model of distributed computations in DVR systems

From the software perspective DVR system is a collection of asynchronous processes p_1, p_2,…,p_n, communicating with each other over the communication network [1]. Without loss of generality assume that every process runs on a separate computer node. Processes have no shared memory and interact with each other solely on the basis of message transfer. Let m_{ij} denote the message sent out by process p_i to process p_j. The transmission delay is finite and unpredictable. The processes do not share global clock that is instantaneously accessible to them. Processes execution and messaging are asynchronous: processes may execute actions at arbitrary time instants and processes don't wait for their successful delivery when sending messages.

In DVR systems there is a differentiation of processes by their function. This can depend on specific task, but there is always differentiation associated with the use of a certain type of communication architecture (see fig. 1). The main types of communication architecture are *client/server* and *peer-to-peer*. There are also mixed architectures, combining these architectures, such as multi-server architecture. Depending on the chosen architecture, *client*, *server* and *peer* processes can be distinguished. In client/server architecture client processes are focused on the individual user's view visualization and state control of his VE object, while server process provides interaction of multiple users. In peer-to-peer architecture peer processes include both functions.

The process interaction is based on a certain high-level protocol. The protocol defines the format of messages and rules of processes interaction. It is usually based on general network protocols, such as TCP/IP protocols.

A *distributed computation* describes the execution of a distributed program by a collection of processes [2]. Process execution can be considered as a sequence of actions, modeled as three types of events: internal events, message send events, and message receive events. Let e_i^l denote l-th event at process p_i. The examples of events in DVR systems are different user's actions in VE, such as objects creation and manipulation, VE state change events, internal service messages etc.

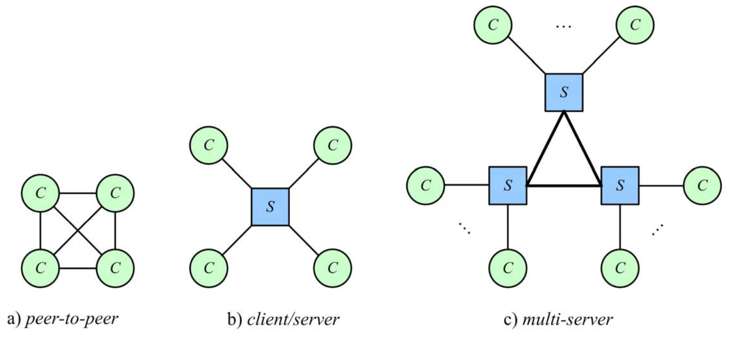

Figure 1. Communication architectures.

The occurrence of events alters the state of the corresponding processes and communication channels, thus causing transitions in the global system state. Internal events change only internal state of the processes in which they occur. A send (or receive) event changes the state of sending (receiving) process and the state of the respective channel. For an arbitrary message m we define the send and receive events as *send*(m) and *receive*(m), respectively.

Local history of process during the computation is a sequence of events $h_i = e_i^1\, e_i^2 \ldots$. It is important that the local history reflects the order in which local events are occurred. Let $h_i^k = e_i^1\, e_i^2 \ldots\, e_i^k$ denote the sequence of events that contains the first k events (h_i^0 corresponds to empty sequence). *Global history* of the computation is a set $H = h_1 \cup \ldots \cup h_n$, containing all the events of processes local histories $h_1, \ldots, h_n$.

In asynchronous distributed systems, where there is no global clock, the events can be ordered only based on cause-effect relation. In other words, two events follow one after another, if the occurrence of one event can affect the outcome of another. Information may flow from one event to another, either because these events belong to the same process, or because the events are of different processes and involved in a message exchange. We define a binary relation on the set H, denoted as $\rightarrow$, that expresses causal dependencies between events in the distributed computation, such that:

1) If $e_i^k, e_i^l \in h_i$ and $k < l$, then $e_i^k \rightarrow e_i^l$,

2) If $e_i = send(m)$ and $e_j = receive\ (m)$, then $e_i \rightarrow e_j$, (1)

3) If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$.

For certain events of the global history there may be no causal dependency. This means that for two such events e и e' both relations $e \rightarrow e'$ and $e' \rightarrow e$ are possible. Such events are said to be concurrent and the relation is denoted as $e \parallel e'$. Formally, distributed computation can be thought of as partially ordered set defined by a pair (H, →). Fig. 2 shows the space–time diagram of a distributed execution involving three processes.

Now we introduce the concept of the *local state* of a process. In general, the local state of a process at any instant of time includes the set of its local variables and the set of messages that are in transit in communication channels incident to the process. Let s_i^k denote a local state of p_i immediately after the occurrence of event e_i^k, and let s_i^0 describes the initial state before any events are executed. Then tuple of local states $S = (s_1, \ldots, s_n)$ represents the *global state* of a

DVR system.

An important issue is what should be stored in the global state. Since distributed computations in DVR system are focused on modeling and visualization of a virtual environment, the global state should reflect the state of the VE. We call this global state as *global virtual environment state*. The virtual environment is a collection of virtual objects with certain sets of attributes. These attributes determine the properties and behavior of each object, and together form an *object state*. Accordingly, the *VE state* is a tuple of all states of its objects.

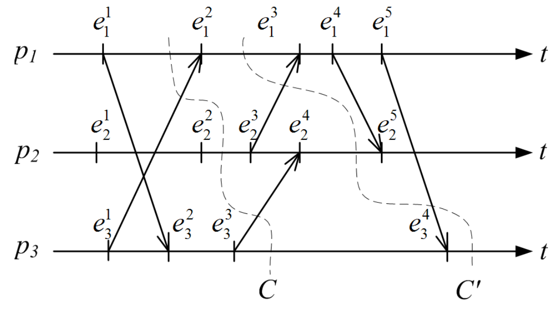

Figure 2. Space-time diagram of a distributed computation.

A cut of a distributed computation is a subset C of its global history H containing all the events of processes local histories to the particular time moments on each process timeline. Cut is defined as $C = h_1^{c_1} \cup \ldots \cup h_n^{c_n}$, where ($c_1,\ldots,c_n$) is a tuple of natural numbers corresponding to the index of last event included for each process. Each cut reflects the global state $S_C = (s_1^{c_1},\ldots,s_n^{c_n})$ of the system. All the events that are before the cut have caused system transition into the state S_C, in turn, all the events occurred after the cut are in dependence on the current system state S_C. A set of events $(e_1^{c_1},\ldots,e_n^{c_n})$ immediately preceding the cut line is called the *frontier* of the cut. At the space-time diagram a cut is represented as a zigzag line joining one arbitrary point on each process line (see cuts C and C' in Fig. 2).

Also let us introduce the notion of distributed computation *observation*. Observation O_i is an ordered sequence of events which comes to the input of the process-monitor during distributed system operation. The process-monitor is a special type of process, which is able to receive all the events taking place in the system. It can be used for recording the history of computing. We also note that in DVR systems, users' (client) processes are both participants of the distributed computations and monitors watching the events occurring in the global VE state.

3. Consistency in DVR systems

3.1. Causal consistency

Causal model of consistency is derived directly from the event-based model of distributed computation considered in the previous section. As discussed previously, with the occurrence of events, DVR system is sequentially advances from one state to another. Causal consistency assumes that the instantaneous snapshots (cuts) of the system state of arbitrary points in time should not violate cause-effect relation between the actions of the various processes. In other words, an effect should not be present without its cause, for example, a message cannot be received if it was not sent. However, it is impossible to capture states of all processes strictly at the same instant due to delays in messages transmission, the differences in processes execution speeds, the absence of global clocks and other reasons. Therefore, for characterizing global state in distributed systems, instead of using particular time instants, cuts are used. In order to meet cause-effect condition such cuts must be *consistent*. The cut C is called consistent if:

$$\forall e, e' : (e \in C) \wedge (e' \rightarrow e) \Rightarrow e' \in C. \qquad (2)$$

Interpreting this expression graphically, all messages received before cut, should be sent before cut as well. In turn, the messages crossing the cut should be sent before the cut and received after the cut. Cut C of fig. 2 is consistent, while cut C' is inconsistent. Consistent cut corresponds to a consistent global state. Condition 2 is also called *causal precedence*.

But being distributed system in the consistent state does not allow to speak about full causal consistency. One more condition, called *causal delivery* should be met:

$$send_i(m) \rightarrow send_j(m') \Rightarrow receive_k(m) \rightarrow receive_k(m'), \qquad (3)$$

for all messages m, m', sending processes p_i, p_j and destination process p_k (this condition also includes the case where messages m, m' are sent from the same process, i.e. $i = j$). This rule ensures that two causally related messages will be received at the destination process in the same order that they were sent.

Thus, DVR system meets causal consistency requirement if for its computation history H system causal precedence (2) and causal delivery (3) conditions are satisfied. In distributed systems model of causal consistency is a means to identify the consistent state of distributed system out of overall flow of events, as well as to identify violations of the cause-effect relationships between events.

3.2. Observational consistency

Observational consistency allows to characterize how good the sequences of global states observed by different users correspond to each other. In terms of considered model, this means that the sequences of events relevant to observations O_1, O_2, …, O_n, captured at monitors p_1, p_2,…, p_n, must be identical to each other and agreed with the global history of computation H.

In practice, because of asynchronous nature of computation in DVR systems, it is impossible to ensure the absolute observational consistency and it can be said only about partial correspondence of sets O_1, O_2, …, O_n. However, the situation can be improved in case of using reliable communication channels that provide delivery of messages in proper order (i.e. satisfying the condition 3), as well as with using of a clock synchronization mechanisms.

3.3. Space-time consistency

Another approach to determining consistency is based on evaluating the conformity of local VE state copies (replicas) at different processes for particular instants or time intervals. While this approach is similar to one of the observational consistency it is used not to specify the order in which global states change each other but to compare the observed states themselves. Also this approach is user related, i.e. it considers VE state from the standpoint of user perception. To characterize the VE state observed by a user a *view* notion is introduced. View is a rendered image of VE observed from the current user's position in the virtual space.

In order to ensure that all users have a sense of presence in a common shared virtual world they should be able to see the same VE state at any instant. In other words, views have to be consistent (not identical, because views could be observed from different viewpoints). This, in turn, means that users' processes should have the same data about current VE objects' states. Objects of the VE can be static (motionless) and dynamic (moving on certain laws, including user controlled objects). When evaluating the consistency of VE views of different users, first of all, it makes sense to consider dynamic objects.

The considered approach implies the introduction of metrics which allow to directly measure the error in determining states of objects. For example, Diot and Gautier in [3] used as a metric the instantaneous distance between remote and local positions of the object, which they called *drift distance*. A more interesting metric later was suggested by Zhou et al in [4] to evaluate how inconsistency can affect the perception of the user in DVR systems. Their metric called *time-space inconsistency* is based on the observation that users make decision after consideration of both the positions and the duration the entities remain in these positions. If this duration is less then human visual perception time, then users may not notice the presence of inconsistency; if the duration is large enough, then users notice the situation and may try to influence it. Based on these considerations, the proposed metric is as follows:

$$\Omega = \begin{cases} 0, \text{if } \left|\Delta(t)\right| < \varepsilon, \\ \int_{t_0}^{t_0+\tau} \left|\Delta(t)\right| dt, \text{if } \left|\Delta(t)\right| \geq \varepsilon, \end{cases} \tag{3}$$

where $\Delta(t)$ is the spatial difference between an object's local true position and its estimation at a remote node; ε is the minimum distance between entities in a VE that a participant can discern; t_0 is the instant at which the difference starts, and τ is the duration for which the difference persists.

In addition, there are other approaches in defining the space-time consistency. Thus, in [5] it was shown that in determining view consistency not only the position of moving objects has great importance, but also the relative objects positions. Also it was demonstrated that when considering complex forms of movement, such as a movement with a variable acceleration, it makes sense to use metrics that are based on derivatives of higher orders, such as speed and acceleration.

4. Proposed consistency model

4.1. Selective consistency principle

Depending on the type, as well as the individual characteristics of the VE objects, their states may include a variety of attributes. For example, the state of static object can be composed of the coordinates of the object, its size, shape, level of detail of polygonal mesh, etc. The states of dynamic objects in addition may include speed, acceleration, the values of forces and moments. Accordingly, to ensure the consistent users interaction, consistency maintenance on all these parameters is required.

As interaction of users in DVR systems is carried out in real time, update and, in particular, rendering of the VE state should be performed with high frequency (for example, in military simulations — no less than 60Hz). Since all the data between processes transmitted over the network, the transfer of complete objects states with such frequency would require a very high bandwidth which is unacceptable when using WAN network. In addition, a complete state of the object is often redundant, and needed only on the side of the process directly managing the object state for accurate state simulating. In most cases for the rest of the processes there is no need in simulating accurate object state and a simplified state model can be used. Thus, other processes do not need in passing specific object characteristics. For example, for dynamic objects there is no need to transfer the values of forces and moments — one can transmit the speed and acceleration of objects instead. The main idea of selective consistency principle is to distinguish the most essential for the specific task attributes of object state and to maintain consistency explicitly on them. These attributes can be selected on the basis of different assumptions. The most important of them is human perception. The selective consistency principle is a relaxation of absolute consistency notion and supposes that there can be inconsistencies in objects states but they are negligible for the user. It allows to find trade-off between consistency and hardware limitations [6,7].

In total, the main features of the selective consistency principle are:

- reducing the number of object state attributes that are needed for the replication;
- use of simplified VE objects models on the processes that are not owners of objects states; it involves state prediction techniques, such as dead reckoning (see [7,8]).

Other issues addressed in the proposed model:

- use of a distributed scene graph to store the VE state;
- data replication strategies;
- use of clock synchronization mechanisms;
- object ownership management (defining ways to access and modify objects states);
- handling of objects interactions.

Below we look at some of them.

4.2. Data replication strategies and distributed scene graph

There are three main data replication strategies: *centralized*, *replicated* and *distributed*. When using a centralized architecture, all VE state data are stored on a dedicated process. In a replicated architecture each process keeps a copy of the entire VE state. In a distributed architecture VE state is distributed among several processes. In our consistency model none of these strategies is used in its original form. Instead, we use hybrid strategy. On the one hand, there is centralized data storage, located on a dedicated server or group of servers, where the most relevant VE state is stored. On the other hand, data from the central storage are replicated on the client processes to reduce the number of requests to a central storage and to increase data access speed (or, in other words, to increase *responsiveness* of DVR system [5,6]). In addition, in case of large and extensive virtual world it makes sense to replicate on the clients not a complete VE state, but its separate parts. The other parts of virtual world can be swapped, when necessary, with the movement of a particular user in the VE. This approach allows to significantly improve the scalability of DVR system. However, for its effective implementation a special form of VE state representation is required. For our purposes it is convenient to organize data in a hierarchical structure which is shared between users, called *distributed scene graph* (*DSG*) [9].

DSG establishes logical and spatial relationships between VE objects. It is headed by a top-level root node which encompasses the whole virtual world. The world is then broken down into a hierarchy of nodes representing either spatial groups of objects, settings of the position of objects, animations of objects, or definitions of logical relationships between objects. The leaves of the graph represent the VE objects themselves, their drawable geometry and their material properties. Formally speaking, a scene graph can be defined as a directed acyclic graph with a random number of children. Main benefits of DSG include:

- fits well with the object structure of the VE and is a natural representation of the virtual world for the human perception;
- provides means for effective 3d-rendering of VE state by implementing various graphics acceleration algorithms (such as *culling* and *level-of-detail techniques*), and collision detection algorithms;
- increases the scalability of DVR systems by distributing the computational load on VE state processing between the processes;
- increases flexibility in data replication;
- simplifies VE objects state management;
- easy programmable.

4.3. Clock synchronization

Clock synchronization is the technique of ensuring that physically distributed processes have a common notion of time. Clock synchronization assumes availability of a *reference clock* in the system that is stored on a dedicated process called *time server*. In our model, each newly connected client synchronizes its clock with the server clock using a special procedure, such as Cristian algorithm [10] or widely used Network Time Protocol [11]. Reference clock set the course of the internal *model time* which is common for the entire DVR system. All events in the VE are related to particular instants of model time. Each global VE state is calculated based on model time.

5. Conclusion and future work

In this paper the applicability of event-based model for distributed computation formalization in DVR systems has been shown. Also, different approaches to consistency definition in such systems were covered.

In order to organize consistent interaction of many users in distributed virtual reality one should not be confined to the specific consistency model. Each model is important and should be taken into account when designing the final system.

The proposed consistency model will be used as a basis for a software framework allowing to create DVR systems for specific application areas. Many issues addressed in the model have not been discussed yet, such as object ownership management or handling of objects interactions, but we will return to them in the near future.

Acknowledgements

This work was supported by the State Analytical Program “The higher school scientific potential development” (project no. 2.1.2/6718).

References

[1] A. Kshemkalyani and M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, New York, NY, USA, 2008.

[2] O. Babaoglu and K. Marzullo, “Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms”, Distributed Systems, Addison-Wesley, Wokingham, 1993, pp. 55-96.

[3] C. Diot and L. Gautier, “A Distributed Architecture for Multiplayer Interactive Applications on the Internet”, IEEE Networks magazine, Vol. 13, 1999, pp. 6-15.

[4] S. Zhou, W. Cai, B. Lee, and S. J. Turner, “Time-Space Consistency in Large-Scale Distributed Virtual Environments”, ACM Trans. Model. Comput. Simul. Vol. 14, 1, 2004, pp. 31-47.

[5] V. Y. Kharitonov, “An Approach to Consistent Displaying of Virtual Reality Moving Objects”, Proceedings of 3rd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008, IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 390-397.

[6] J. Smed, H. Hakonen, Algorithms and Networking for Computer Games, UK, Chichester: John Wiley & Sons, 2006.

[7] V.Y. Kharitonov, “Methods of efficiency enhancement of network interaction in distributed systems of virtual reality”, Proceedings of 2nd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2007, IEEE Computer Society, Los Alamitos, CA, USA, 2007, pp. 305-308.

[8] S. Singhal, M. Zyda, Networked virtual environments: design and implementation, ACM Press/Addison-Wesley Publishing Co., New York, NY, 1999.

[9] M. Naef, E. Lamboray, O. Staadt and M. Gross, “The blue-c distributed scene graph”, Proceedings of the Workshop on Virtual Environments 2003, EGVE '03, Vol. 39. ACM, New York, NY, 2003, pp. 125-133.

[10] F. Cristian, “Probabilistic clock synchronization”, Distributed Computing, Vol. 3, 1989, pp. 146-158.

[11] D. L. Mills, “Internet time synchronization: the network time protocol”, IEEE Transactions on Communications, 39(10), 1991, pp. 1482-1493.