ISSN : 0974 - 7435 Volume 10 Issue 18

2014

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(18), 2014 [10717-10723]

A real-time causal order delivery approach in large-scale DVE systems

Zhongli Liu[1], Hangjun Zhou[1,2,*], Sha Fu[1], Guang Sun[1], Wei Zou[1]
[1]Department of Information Management, Hunan University of Finance and Economics, Chang Sha, Hu Nan, 410205, (P.R.CHINA)
[2]School of Mechanical Engineering, Nanjing University of Science and Technology, Nan Jing, Jiang Su, 210094, (P.R.CHINA)
E-mail : zhjnudt@gmail.com

ABSTRACT

The aim of a Distributed Virtual Environment (DVE) system is to simulate the scenarios of the real world and offer the sufficient fidelity with real-time property. However, as more and more large-scale DVE systems applied on asynchronous Wide Area Network (WAN), the traditional causal order control methods could not function well due to the large and dynamic network transmission latency, therefore, how to maintain the causality of events in real time becomes the new challenge of this research field. In this article, a novel event ordering control method is proposed based on the message transmission time interval and event arrival time range calculation. This approach provides algorithms and procedures to effectively select the causal control information dynamically adapted to the network latency and irrelevant to the computing node scale. The evaluation results of groups of experiments demonstrate that the approach is more efficient in preserving causal order delivery of events in large-scale networks, and in the meanwhile, meets the real-time constraint of a DVE.

KEYWORDS

Distributed virtual environment; Causal order; Real-time Constraint; Large-scale asynchronous network system.

INTRODUCTION

A DVE system is a computer-generated three-dimensional virtual world where multiple participants could exchange data simultaneously and interact immersively with each other[1,2]. The nodes in a DVE don't share any memory and communicate merely by message transmitting[3]. Furthermore, when a DVE is applied in wide area network, it is usually hard for all the nodes to have a common simulation clock[4]. In order to make a DVE system resembling our living world, not only the cause-effect logic order of events should be preserved correctly, but also the causality should be maintained under real-time constraint due to the real-time property of events happened in real world.

As for the traditional methods in related works, Vector Times[5] and Immediate Dependence Relation (IDR)[6] are the two classical categories of the traditional asynchronous causal order control approaches. However, as these approaches usually ignore the time validity of event and assume the events can always arrive in time, it is hard for them to reduce the causality violations with real-time property. A Δ-causal order method is proposed[7], which has taken the time validity of event into consideration. Nevertheless, this method requires that the simulation time of each node is accurately synchronized and each event must have the identical valid time "Δ", which is not suitable for the asynchronous DVEs.

In order to satisfy the real-time constraint on the delivery of causal order in a DVE with asynchronous simulation clocks, we analyze the root of the problem and propose a novel event ordering control approach based on the message transmission time interval and event arrival time range calculation in this paper. This method provides the way to select the causal control information dynamically adapted to the network latency and irrelevant to the system scale. Experimental results show that our method is more efficient in preserving causal order delivery of events, and in the meanwhile, meets the real-time constraint of a DVE.

The rest of the paper is organized as follows. The real-time casual order delivery is defined in section 2. Then, message transmission time interval is described in section 3. A new causal order approach is proposed in section 4 and groups of experiments are implemented in section 5. Conclusions are stated in section 6.

REAL-TIME CAUSAL ORDER DELIVERY

Assume that the set of all nodes in a DVE is $P = \{p_1, p_{2,}.., p_n\}$. For any event e_x generated at p_i, we use $r(e_x) = (i, a)$ as its identification such that i is the number of the node and a is the logical time when e is generating[3.4]. t^i_x is set to describe the simulation time p_i can identify e_x, i.e., if e_x is generated at p_i, t^i_x indicates the time when e_x is generated or if e_x is not generated but received at p_i, t^i_x indicates the time when e_x is receiving.

In order to maintain the causal order consistency, the causality control approaches need to add the relevant causal control information in the message containing an event. For instance, if e_y is generated at p_j and e_x is the any cause of e_y, p_j needs to select the causal control information $CI(e_y)$ before it sends e_y. Both e_y and $CI(e_y)$ would be added into the same message M.As for the receiving node(s) p_{des}, it might happen that when the valid time e_y is terminated, some the causes of e_y couldn't arrive p_{des} yet for the large network transmission delay. In this case, it is required that p_{des} could identify the received causes of e_y with utilizing $CI(e_y)$ and preserve the causality of received events. Thus, the real-time causal order delivery of events in a DVE system could be defined as follows.

Definition1 (Real-time Causal Order Delivery). $\forall p_{des} \in P$, the real-time causal order delivery of events at p_{des} is preserved iff

$\forall e_y \in V_{des} - H_{des}$, at the time when the lifetime of e_y is terminated or e_y is required to be delivered, $\forall e_x \in V_{des} - H_{des}$, if $e_x \to e_y$, e_x must be delivered before e_y at p_{des};

$\forall e_y \in V_{des} - H_{des}$, let $F_y = \left\{ e_x \middle| \forall e_x \in E \wedge e_x \to e_y \right\}$, if the lifetime of e_y is still vaild and $F_y \subseteq H_{des}$, e_y must be delivered immediately at p_{des}.

Where V_{des} denotes all the events that have been received at p_{des}, and H_{des} denotes all the events that have been delivered at p_{des}, and E denotes all the events generated at P.

MESSAGE TRANSMISSION TIME INTERVAL

To select effective control information, we can predict the round trip time between p_i and p_j at first with the distributed predicting mechanism[8]. In the application level of a distributed system, it may regard the half value of the round trip time as the message transmission delay time from p_i to p_j, which is denoted as Δt_{ij}.

However, Δt_{ij} merely means the transmission time from p_i to p_j, based on which we can solely calculate the causal control information suitable for p_j. As shown in Figure 1, event e_y is generated at p_j, and sent to p_i and p_k at the moment t^j_y in a message. Respectively, experiencing different transmission time Δt_{ji} and Δt_{jk}, e_y arrives at p_i and p_k. Therefore, p_j can record and update a message transmission time interval: $[\Delta t_{min}, \Delta t_{max}]$, where Δt_{min} indicates the minimal transmission time at current moment and Δt_{max} indicates the maximal transmission time at current moment. With $[\Delta t_{min}, \Delta t_{max}]$, p_j can calculate the arriving range of an event sent to p_{des}. For instance, when p_j sends e_y is at t^j_y, the arriving range of e_y at p_{des} can be denoted as $[t^j_y + \Delta t_{min}, t^j_y + \Delta t_{max}]$. Similarly, the arriving range of e_x sent from p_i is $[t^i_x + \Delta t_{xmin}, t^i_x + \Delta t_{xmax}]$. If e_x is the predecessor event and

e_y is the corresponding successor event, $CI(e_y)$ suitable for all p_{des} to preserve causality between e_x and e_y in real time could be selected through comparing $[t^j_y + \Delta t_{min}, t^j_y + \Delta t_{max}]$, $[t^i_x + \Delta t_{xmin}, t^i_x + \Delta t_{xmax}]$ and $[t^j_x + \Delta t_{xmin}, t^j_x + \Delta t_{xmax}]$.

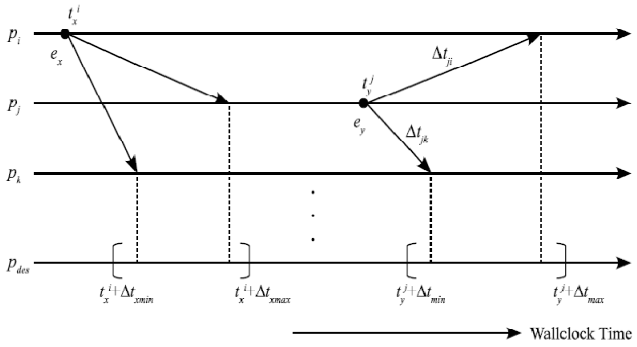

Figure 1 : The message arriving ranges

In order to acquire the correct $[\Delta t_{min}, \Delta t_{max}]$ at the sending moment, p_j needs to maintain and update a network coordinate vector $\{X_1, X_2, .., X_n\}$, where Xi indicates the latest network coordinate of pi. The manner to handle the update message about X_q from p_q, $q = 1,2,...,n$ is stated in Figure 2.

Procedure 1 Update the network coordinate vector

1: **if** $(q = min$ and $\|X_j \quad X_q\|/2 > \Delta t_{min})$ or $(q = max$ and $\|X_j - X_q\|/2 < \Delta t_{max})$ or $q = j$ **then**
2: **for** $a = 1$ to n **do**
3: **if** $a = 1$ **then**
4: $\Delta t_{min} = \Delta t_{max} = \|X_j \quad X_a\|/2$
5: $min = max = a$
6: continue
7: **end if**
8: **if** $a = j$ **then**
9: continue
10: **end if**
11: **if** $\|X_j - X_a\|/2 < \Delta t_{min}$ **then**
12: $\Delta t_{min} - \|X_j - X_a\|/2$
13: $min = a$
14: **end if**
15: **if** $\|X_j - X_a\|/2 > \Delta t_{max}$ **then**
16: $\Delta t_{max} = \|X_j - X_a\|/2$
17: $max = a$
18: **end if**
19: **end for**
20: **else**
21: **if** $\|X_j - X_q\|/2 < \Delta t_{min}$ **then**
22: $\Delta t_{min} = \|X_j - X_q\|/2$
23: $min = q$
24: **end if**
25: **if** $\|X_j - X_q\|/2 > \Delta t_{max}$ **then**
26: $\Delta t_{max} = \|X_j - X_q\|/2$
27: $max = q$
28: **end if**
29: **end if**

Figure 2 : Update the network coordinate vector

NEW CAUSAL ORDER CONTROL APPROACH

In this section, we design and propose a new event ordering control approach to preserve the causal order delivery in real time. The approach majorly contains two distributed and recursive procedures called by a node: to select the causal control information and to deliver received messages.

For event e_y generated at p_j, $CI(e_y)$ is selected through comparing and calculating $[t^j_y + \Delta t_{min}, t^j_y + \Delta t_{max}]$, $[t^i_x + \Delta t_{xmin}, t^i_x + \Delta t_{xmax}]$ and $[t^j_x + \Delta t_{xmin}, t^j_x + \Delta t_{xmax}]$ so as to acquire control information dynamically adapted to network latency and maintain causality of events at p_{des}. The specific procedure to select $CI(e_y)$ is depicted in Figure 3.

Procedure 2 Select the causal order control information

1: **if** the element pointed by cf is not in $CF(p_j)$ **then**
2: return false
3: **end if**
4: **if** $(*cf) \in CI(e_y)$ **then**
5: p_{tmp} = getPointerInCI($cf \rightarrow r(e_x)$)
6: $sp \rightarrow ptr[cn] = p_{tmp}$
7: return ture
8: **end if**
9: $p_{tmp} = CI(e_y)$.add($(*cf)$)
10: **if** $sp \neq null$ and $cn \neq null$ **then**
11: $sp \rightarrow ptr[cn] = p_{tmp}$
12: **end if**
13: **if** $(p_{tmp} \rightarrow (t_x^j + \Delta t_{xmax})) \leq (t_y^j + \Delta t_{min})$ **then**
14: **for** $a = 0$ to $num - 1$ **do**
15: $p_{tmp} \rightarrow ptr[a] = null$
16: **end for**
17: **else**
18: **for** $a = 0$ to $num - 1$ **do**
19: **if** ! (Procedure 2 is recursively called by p_j with the arguments: p_{tmp}, a and $cf \rightarrow ptr[a]$) **then**
20: $p_{tmp} \rightarrow ptr[a] = null$
21: **end if**
22: **end for**
23: **end if**
24: return ture

Figure 3 : Selection of $CI(e_y)$

The variables used in the procedure are described as follows:

$CF(p_j)$: the adjacency list used by p_j to store and updates the causality relation of delivered events. Each element in $CF(p_j)$ has five variables: ($r(e_x)$, t^j_x, Δt_{xmin}, Δt_{xmax}, $ptr[num]$), such that e_x is an predecessor event of e_y and t^j_x is the moment when p_j receives or generates e_x. $ptr[num]$ is an pointer array in which each pointer points to an immediate predecessor element of $r(e_x)$ in $CF(p_j)$. p_j would periodically delete the redundant elements in $CF(p_j)$.

$CI(e_y)$: the adjacency list to store the causal control information of e_y. Each element in $CI(e_y)$ has five variables: ($r(e_x)$, t^j_x, Δt_{xmin}, Δt_{xmax}, $ptr[num]$). Usually, ($r(e_x)$, t^j_x, Δt_{xmin}, Δt_{xmax} could be replicated from the corresponding element in $CF(p_j)$, but the $ptr[num]$ needs to be revalued to point to the immediate predecessor elements of $r(ex)$ in $CI(e_y)$.

The variable sp is the pointer pointing to the element in $CI(e_y)$ which recursively calls the procedure. Variable cn belongs to $[0, num-1]$, cf is equal to $ptr[cn]$ of the element in $CF(p_j)$ which is corresponding to the element in $CI(e_y)$ pointed by sp.

After the selection of $CI(e_y)$, a message containing e_y and $CI(e_y)$ will be sent out by sending algorithm. The sending algorithm calling the procedure of causal order control information selection is demonstrated in Figure 4, where $VT[p_j]$ denotes the logical time of p_j. After the selection of $CI(e_y)$, a message containing ey and $CI(e_y)$ will be sent out.

Message Sending Algorithm

1. update the logical time $VT[p_j] = VT[p_j] + 1$;
2. acquire the current local simulation time t_y^j ;
3. count = the number of the nearest causes of e_y ;
4. while(count>0)
   {
5. call the procedure to select $CI(e_y)$;
6. count--;
   }
7. encapsulate e_y and $CI(e_y)$into message M ;
8. send message M ;
9. add ($r(e_y)$, t_y^j, Δt_{ymin}, Δt_{ymax}, p_cs) into $CF(p_j)$;
10. clear$CI(e_y)$;
11. exit ;

Figure 4 : Message sending algorithm

When p_{des} receives the message M containing e_y and $CI(e_y)$, it is necessary for it to determine whether M should be delivered, discarded or buffered in $MQ(p_{des})$, which is achieved by the message receiving algorithm described in Figure 5. In the receiving algorithm, $MQ(p_{des})$ is the message buffer queue at p_{des}. If the received messages cannot satisfy the condition of causal order delivery, and they are respectively within their own lifetimes and not requested to delivery by their successor message, p_{des} would buffer these messages.

Message Receiving Algorithm

1. acquire the current local simulation time t_y^{des} ;
2. if the valid time of e_y is terminated
   {
3. abandon the message ;
4. exit ;
   }
5. else
   {
6. if there exists an delayed cause
   {
7. buffer the message M with t_y^{des} into $MQ(p_{des})$;
8. exit ;
   }
9. call the procedure to deliver message M with t_y^{des} ;
10. exit ;

Figure 5 : Message receiving algorithm

The line 9 of Figure 5 calls the recursive procedure to deliver a message, which is specifically described in Figure 6.

Procedure 3 Deliver the message M

1: **if** M containing e_y and $CI(e_y)$ arrives late or is expired **then**
2: discard the message
3: return false
4: **else if** the lifetime of M is terminating or Procedure 3 is currently called by M's successor message to deliver M **then**
5: **if** the immediate predecessor messages of M can be identified in $MQ(p_{des})$ **then**
6: Procedure 3 is recursively called to deliver the messages
7: **else if** the predecessor messages having the nearest causality relation with M can be identified in $MQ(p_{des})$ **then**
8: Procedure 3 is recursively called to deliver the messages
9: **end if**
10: **else if** !(all the immediate predecessor messages of M have been delivered) **then**
11: buffer M into $MQ(p_{des})$
12: return ture
13: **end if**
14: **if** M is not delivered **then**
15: $CF(p_{des})$.add$(r(e_y), t_y^{des}, \Delta t_{min}, \Delta t_{max}, ptr[num])$
16: deliver the event e_y
17: **end if**
18: return ture

Figure 6 : Delivery of a receiving message

EXPERIM ENT RESULTS AND ANALYSIS

On purpose of verifying the performance of the event ordering control methods, groups of experiments are conducted on the PC cluster in the key laboratory of distributed and parallel processing in our university. The characteristics of the message transmission latency of wide area network are emulated by a Spirent ConNIE (Converged Network Impairment Emulator) in the cluster laboratory. A peer-to-peer DVE is initiated in which each computing node is configured with asynchronous simulation clock[9,10]. Furthermore, in groups of experiments, an air battle simulation system is developed to run as verification instance with different scales: 2400, 4800, 7200, 9600, 12000. For the simulation clocks of all nodes are asynchronous, the representative causal control methods, Vector Time[5] and IDR[6], are chosen to compare with our method in the experiments.

Figure 7 illustrates the violation numbers of causal order delivery of the simulations with different scales. Due to the inadequate control information, the violation numbers of IDR are the most when there exist lately arrived events. Due to the large message transmission amount, the violations of Vector Time under time constraint occurs more than those of our method in each scale.

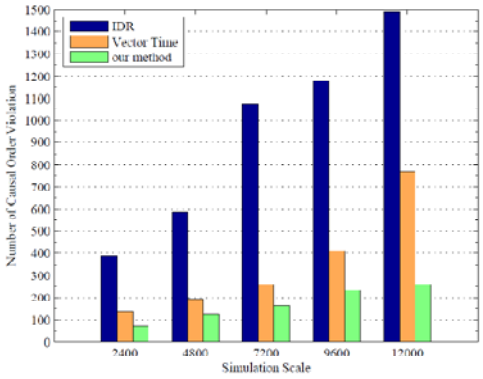

Figure 7 : Violation numbers of causal order

Figure 8 demonstrates the percentages of the average amount of control information of our method compared with Vector time under different network transmission conditions in each scale. As we can see, in scale 2400, the percentage under 200ms mean network latency is less than 26% and the percentage under 50ms latency is lower than 5%. As the scale expands, the percentages under all conditions decrease significantly, which indicates that the message transmission amount of our method is far less than that of Vector time.

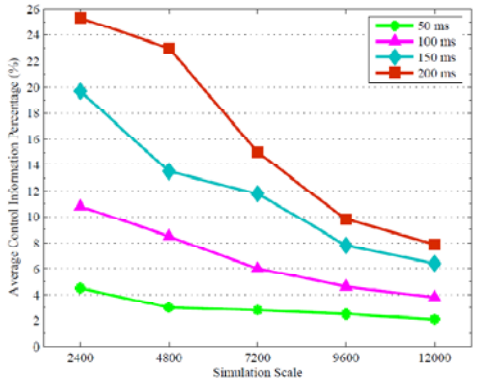

Figure 8 : Percentage of average amount

Figure 9 demonstrates the average message delivery time costs of the three approaches in different scales. The cost of our approach is low and could meet the real-time constraint with the small overhead of transmission and computation.

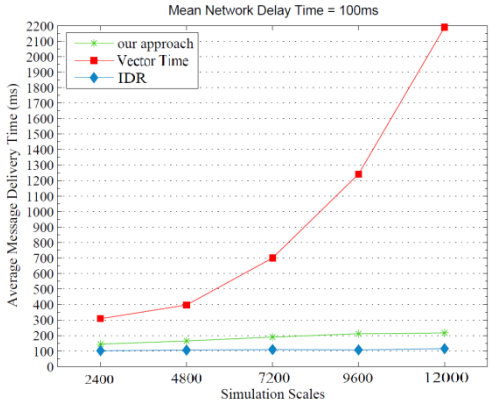

Figure 9 : Average message delivery time

CONCLUSIONS

With the rapid development of Internet and mobile technologies, more and more large-scale distributed virtual environments (DVEs) are implemented on the asynchronous WAN, where the real-time causal order preservation becomes a new and important issue. In this article, we present a novel control approach with message sending and receiving algorithms

calling two recursive procedures to select the causal control information adapted to the network latency and deliver received messages with real-time constraint. Experiments demonstrate that our approach could effectively reduce the numbers of causal order violations in real time and decrease the average amount of control information overhead.

ACKNOWLEDGEMENTS

This research work is supported by the Hunan Science and Technology Project (No. 2014GK3042) and Scientific Research Fund of Hunan Provincial Education Department (13C093). It is also supported by the Hunan Science and Technology Project (No. 2012FJ6011) and the Construct Program of the Key Discipline in Hunan Province, China.

REFERENCES

[1] M.Fujimoto; Parallel and Distributed Simulation Systems [M]. New York: Wiley Interscience Press, **(2000)**.

[2] Z.Wang Zhangh; A Run-Time Infrastructure Based On Service-Distributed Architecture [J]. Applied Mathematics & Information Sciences **7(2)**, 595-604 **(2013)**.

[3] M.Raynal; Distributed Algorithms For Message-Passing Systems [J]. Springer-Verlag Press, Berlin **(2013)**.

[4] L.Lamport; Time, Clocks, And The Ordering Of Events In A Distributed System [J]. Communications of The Acm **21(7)**, 558–565 **(1978)**.

[5] R.Schwarz, F.Mattern; Detecting Causal Relationships In Distributed Computations: In Search Of The Holy Grail [J]. Distributed Computing **7(3)**, 149-174 **(1994)**.

[6] P.Hernndez, J.Fanchon; The Immediate Dependency Relation: An Optimal Way To Ensure Causal Group Communication [J]. Annual Review of Scalable Computing **6(3)**, 61-79 **(2004)**.

[7] R.Baldoni, M.Raynal; Efficient Δ-Causal Broadcasting [J]. International Journal Of Computer Systems Science And Engineering, **13**, 263-271 **(1998**).

[8] F.Dabek, R.Cox; Vivaldi: A Decentralized Network Coordinate System [C]. In Proc. of Sigcomm Conf., 426-437 **(2004)**.

[9] J.Chimal-Eguía, A.Chavez-Valle; Modelling the Use of Programming Languages: A Very Simple Approach [J]. Applied Mathematics & Information Sciences, **8(3)**, 1037-1040 **(2014)**.

[10] C.Bouras, E.Giannaka; A Simulation Modelling Tool for Distributed Virtual Environments [J]. Simulation Modelling Practice and Theory, **25**, 1-16 **(2012)**.