ISSN 1000-1239/CN 11-1777/TP
48(9): 1768-1780, 2011

异步 DVE 系统中生命周期约束下的因果一致性控制方法研究

周航军[1,3] 张 伟[2] 彭宇行[1] 李思昆[2]

[1](并行与分布处理国防科技重点实验室(国防科学技术大学) 长沙 410073)

[2](国防科学技术大学计算机学院 长沙 410073)

[3](湖南财政经济学院信息管理系 长沙 410205)

(zhouhangjun@nudt.edu.cn)

A Lifetime-limited Causal Order Control Method in Asynchronous DVE System

Zhou Hangjun[1,3], Zhang Wei[2], Peng Yuxing[1], and Li Sikun[2]

[1](*Key Laboratory of Science and Technology for National Defence of Parallel and Distributed Processing* (*National University of Defense Technology*), *Changsha* 410073)

[2](*College of Computer*, *National University of Defense Technology*, *Changsha* 410073)

[3](*Department of Information Management*, *Hunan University of Finance and Economics*, *Changsha* 410205)

Abstract Distributed virtual environment(DVE) is a computer-generated virtual space that simulates the real world. Therefore, in a DVE, causal order consistency is required to be preserved in real time, which means the causal events must be delivered within the lifetime of the result event. However, due to the network latency, part of causal events may not arrive the receiving node in time especially in large-scale DVE, and then the causality between the arrived causal events and the result event can not be maintained within the lifetime of it. In related work, some do not consider the causality with lifetime limitation based on the presumption that all events can arrive in time, while others require accurate synchronous simulation clock and their control overhead is closely-coupled with the system scale so that causal control efficiency becomes very low in large-scale DVE. In this paper, we propose a novel lifetime-limited causal order (LCO) control method that can compare asynchronous time of different nodes, conclude the ending condition of multi-path causal order control information selection and dynamically adapt the causal control information according to network latency variation. Thus even when part of causal events can't arrive in time, the causality among arrived events can be preserved within lifetime limitation using causal control information selected by LCO. The experiment results demonstrate that LCO can effectively preserve causal order consistency within lifetime and the overhead of the causal control information is irrelevant with system scale.

Key words large-scale distributed virtual environment; lifetime-limited; causal order consistency; causal order control information; asynchronous clock

摘 要 分布式虚拟环境是模拟现实世界的虚拟空间,对因果一致性控制具有实时性要求,必须在事件生命周期结束前得到维护.然而,在大规模网络条件下,网络传输高延迟和动态性会导致部分事件不能及时到达,使已传到事件间因果关系无法在生命周期限制内有效传递.在现有方法中,部分方法基于所有事件一定能及时传到的假设,没有考虑生命周期对因果关系的制约;而另一部分方法虽然考虑了生命周期的约

收稿日期:2010-03-29;**修回日期**:2011-05-24

基金项目:国家"九七三"重点基础研究发展计划基金项目(2011CB302601);湖南省自然科学杰出青年基金项目(S2010J5050)

束,但其因果关系传递要求仿真时钟精确同步,且因果控制效率随系统规模的扩大而快速降低,限制了虚拟环境的普适性和实时性.提出了生命周期约束下的因果一致性控制方法 LCO,突破了异步时钟间的时间值比较、多路径因果控制信息选择的终止条件、网络状况敏感的因果控制信息动态调节等关键技术,能够在事件无法及时传到时,仍可以根据已传到的事件计算出因果传递关系.实验证明,LCO 既能维护生命周期内的因果一致性,又使因果控制信息量与系统规模无关,降低网络传输和计算开销.

关键词 分布式大规模虚拟环境;生命周期限制;因果一致性;因果控制信息;异步时钟

中图法分类号 TP391

在大规模分布式虚拟环境(distributed virtual environment, DVE)中,事件间因果关系一致性要得到有效控制,各节点就必须维护原因事件执行后再处理结果事件的执行顺序.然而,分布式虚拟环境是对真实世界的模拟,消息接收和事件处理存在实时性要求,必须在一定时间内完成,事件的等待和执行受到其生命周期约束.在广域网运行的大规模虚拟环境中,网络传输的高延迟和动态性会导致部分原因事件无法在结果事件的生命周期内及时传到,如果不有效处理将会破坏已到达事件间的因果关系.因此,如何在事件的生命周期内有效维护因果一致性已成为广域网分布式大规模虚拟环境中亟需解决的重点和难点问题[1-5].

向量时间法[6-9]和直接依赖关系法[10-13]是解决因果一致性问题的传统控制方法,但它们没有考虑事件的生命周期限制,认为事件一定能及时传到,因此难以有效实时控制由传输迟到所导致的因果违背情况.文献[3]给出了一种带生命周期限制的"Δ-因果序"控制方法,要求各节点仿真时间精确同步且每个事件消息都拥有相同的生命周期值,降低了虚拟计算环境的普适性和实时性.文献[4]提出了"生命周期限制下的因果序"控制方法,可以为不同消息设置不同生命周期值,但由于该方法将整个向量时间作为因果控制信息附加到每个消息中,导致传输和计算开销与系统规模紧密相关,可扩展性差,随着节点规模数的增大会造成响应时间长、因果违背增多等问题.

在事件生命周期的约束下,维护因果一致性是分布式大规模虚拟环境的挑战性问题:①时钟不同步造成不同节点上的事件时间值不可比;②生命周期的约束可能造成原因事件在结果事件生命周期内无法及时传到;③传输带宽的限制导致消息携带的因果控制信息量不能过大.针对该问题,本文提出了一种新的带生命周期限制的因果一致性控制方法 LCO(lifetime-limited causal order),其基本思想是基于先发后收的物理常识来确定不同节点上时间值的比较原则,突破了事件时间值比较的关键技术;通过适当选择因果控制信息内容,确保任一结果事件可以依据其因果控制信息,在已传到的事件中计算出可重构直接依赖关系的非直接原因事件,不受未到达事件的影响;根据网络状况动态调节因果控制信息量,既能够维护因果关系,又不占用过多的网络资源.实验表明,该方法能够通过有限的因果控制信息,维护生命周期限制下的因果一致性.

1 问题描述

设分布式虚拟环境全体仿真节点集合为 $P=\{p_1,p_2,\cdots,p_i,\cdots,p_n\}$,集合内的节点互为对等节点.对其中任一节点 p_i 上产生的事件 e,用 $r(e)$ 作为其标识,$r(e)=(i,a)$,i 为节点编号,a 为 p_i 产生 e 的逻辑时间[7,14],通过 $r(e)$ 可唯一识别 e.对于事件 e,用 t_e^i 表示 e 被节点 p_i 所感知的时间,若 p_i 是 e 的产生节点,则 t_e^i 表示 e 在 p_i 上的产生时间;若 p_i 是 e 的接收节点,则 t_e^i 表示 e 在 p_i 上的接收时间.将 p_i 产生事件集合记为 R_i,p_i 接收事件集合记为 V_i,p_i 已处理事件集合记为 H_i,由于因果不一致现象只在节点之间的事件间发生,因此,P 上的全部事件集合记为 $R=\bigcup_{i=1}^{n}R_i$.节点间通过发送和接收消息 M 来传递事件.在上述基础上传统因果一致性定义描述如下:

定义 1. 发生在先关系[14-15].设 $e_x\in R_i$,$e_y\in R_j$,$r(e_x)=(i,a)$,$r(e_y)=(j,b)$,则 e_x 与 e_y 的发生在先关系可描述为

① $e_x\rightarrow e_y$,如果满足 $i=j$ 且 $a<b$;

② $e_x\rightarrow e_y$,如果 $i\neq j$,且 e_x 为发送 M 的事件,e_y 为接收 M 的事件;

③ $e_x\rightarrow e_y$,如果 $\exists e_z\in R$,满足 $e_x\rightarrow e_z$ 且 $e_z\rightarrow e_y$.

若 e_x 与 e_y 满足上述情况之一,称 e_x 先于 e_y 发生,记为 $e_x\rightarrow e_y$.发生在先关系定义了分布式系统

不同事件间的先后顺序,在分布式虚拟环境中被作为判断事件间因果关系的规则. 如果 $\neg(e_x \rightarrow e_y)$ 且 $\neg(e_y \rightarrow e_x)$,则称 e_x 与 e_y 为并发关系,记为 $e_x \parallel e_y$.

定义 2. 直接依赖关系[10-11]. 设 $e_x \in R_i, e_y \in R_j$, e_x 与 e_y 满足直接依赖关系当且仅当 $e_x \rightarrow e_y$ 且 $\forall e_z \in R, \neg(e_x \rightarrow e_z \rightarrow e_y)$ 成立,记为 $e_x \downarrow e_y$.

若 $e_x \downarrow e_y$,表示 e_x 是 e_y 的因果关系中最接近 e_y 的原因事件,称为 e_y 的直接原因事件,通常以直接原因事件被执行作为处理结果事件的触发条件.

定义 3. 因果一致性[7,15]. 设 $e_x \in R_i, e_y \in R_j$,若 $e_x \rightarrow e_y$ 成立,则 $\forall p_k \in P$,如果 p_k 需要既处理 e_x 又处理 e_y,则 p_k 一定先处理 e_x 再处理 e_y.

定义 4. 消息生命周期. 消息生命周期由消息发送方指定,其值表示消息发送后可持续的未被处理的最长物理时间间隔,记为 ΔT.

分布式虚拟环境的实时性要求使得事件的活跃期受到其生命周期的限制,即消息被发送后的 ΔT 内其数据对接收节点有意义. 而广域网高延迟可能导致原因事件消息无法都在结果事件生命周期内传到,这将使已传到的间接原因事件转变为并发事件,出现因果违背情况. 为了减少因果不一致现象,实时传递因果关系需要将未及时传到的事件视为失效事件[3-4],提出带生命周期约束的因果一致性,保证已传到事件间的正确因果关系.

定义 5. 生命周期约束下的因果一致性. $\forall p_i \in P$,如果 p_i 上事件处理满足下述条件:

① $\forall e_y \in V_i - H_i$,在 e_y 生命周期 ΔT 结束时,对 $\forall e_x \in V_i - H_i$,若 $e_x \rightarrow e_y$,则 P_i 一定先处理 e_x 再处理 e_y;

② $\forall e_y \in V_i - H_i$,记 $F_y = \{e_x \mid \forall e_x \in R \wedge e_x \rightarrow e_y\}$,在 e_y 生命周期 ΔT 内,$F_y \subseteq H_i$ 成立时,则 P_i 一定立即处理 e_y.

则称 p_i 维护了生命周期约束下的因果一致性,记为 $e_x \xrightarrow{\Delta T} e_y$.

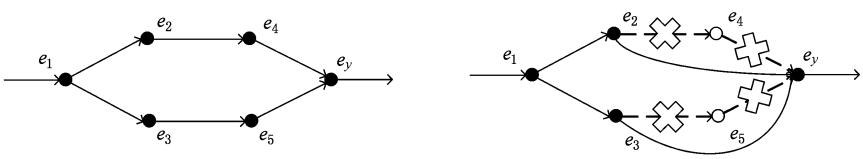
分布式虚拟环境的实时性要求使因果一致性维护受到事件生命周期约束. 如果按定义 3 来维护因果一致性,即基于"结果事件在生命周期结束前其所有原因事件均能被接收并执行完毕"的假设来处理,则为了能等到所有原因事件,需要将结果事件生命周期值设置得足够大,但网络高延迟会造成后续事件被长时间阻塞,破坏虚拟环境的实时性. 因此,需要提出实时生命周期约束下有效维护因果一致性的控制方法.

2 生命周期约束下的因果一致性控制方法 LCO

2.1 设计思想

在分布式控制方法中,为维护因果一致性,必须在传输事件的同时附加其因果控制信息. 例如,若 $e_y \in R_j, e_x \rightarrow e_y$,则节点 p_j 在发送 e_y 之前需要为 e_y 选取相应的因果控制信息 $CL(e_y)$,包括原因事件标识 $r(e_x)$ 等内容,并将 $CL(e_y)$ 添加到 e_y 的消息 M 中发送出去(有时也将发送消息 M 简述为发送事件 e_y). 对事件 e_y 的接收节点 p_{des} 来说,网络传输延迟可能导致在 e_y 生命周期结束时直接原因事件还没有传到. 这时,p_{des} 要能够利用 $CL(e_y)$,从 V_{des} 识别出 e_y 的可重构直接依赖关系的非直接原因事件,保持已传到事件间因果关系的传递性.

图 1(a)给出了在节点 p_j 上 $e_1, e_2, e_3, e_4, e_5, e_y$ 等事件间的因果关系. p_j 会把 e_y 和 $CL(e_y)$ 封装在消息 M 中一同发送出去. p_{des} 通过解析 $CL(e_y)$ 来判断 e_y 的哪些原因事件已经传到,因为 $e_4 \downarrow e_y$ 且 $e_5 \downarrow e_y$,所以 p_{des} 应尽可能以 e_4, e_5 传到并被执行作为处理 e_y 的触发条件. 若在 e_y 生命周期结束时 e_4, e_5 尚未传到,而到达事件只有 e_1, e_2, e_3,则需要在 p_{des} 上建立如图 1(b)所示因果关系. 为了达到这一目的,要求 p_{des} 能够根据 $CL(e_y)$ 识别出与 e_4, e_5 分别具有直接依赖关系的原因事件,即 e_2, e_3,在它们与 e_y 之间分别建立新的直接依赖关系 $e_2 \downarrow e_y$ 与 $e_3 \downarrow e_y$,使 e_2, e_3 的执行作为处理 e_y 的触发条件. 而对于 e_2, e_3, p_{des} 再分别利用 $CL(e_2), CL(e_3)$ 去识别可作为其触发条件的原因事件. 这样,通过递归维护 $e_x \xrightarrow{\Delta T} e_y$,满足生命周期约束下的因果一致性控制需求.



(a) Causal order relationship of $e_1, e_2, e_3, e_4, e_5, e_y$ on p_j (b) Causal order relationship of e_1, e_2, e_3, e_y on p_{des}

Fig. 1 The causal order relationship among events.

图 1 事件间因果关系示意图

事件间正确的因果关系是通过因果控制信息来维护的,控制信息的选取将影响传输开销和计算开销以及因果一致性维护的性能和功能.因此,有效选取恰当的因果控制信息是解决生命周期约束下因果一致性问题的关键.即在 p_j 为 e_y 选取 $CL(e_y)$ 时,若 $CL(e_y)$ 的内容选少了则当迟到的原因事件增多时,p_{des} 可能无法利用 $CL(e_y)$ 从 V_{des} 中识别出满足 $e_x \rightarrow e_y$ 的 e_x,不能维护 $e_x \xrightarrow{\Delta T} e_y$;如果增加过多 $CL(e_y)$ 信息量,虽然维护 $e_x \rightarrow e_y$ 的能力得到增强,但消息在网络上的传输开销和节点上的计算开销增大,反而更难维护 $e_x \xrightarrow{\Delta T} e_y$.

解决"在高延迟动态变化的网络中,怎样选取恰当量的因果控制信息,有效维护带生命周期限制的因果一致性"这一难题的基本思想是在发送 e_y 前,p_j 主动预测 e_y 的网络传输时间以及传输到 p_{des} 后可能遇到的因果关系断开情况,采用时间点变换策略解除时钟不同步对 $CL(e_y)$ 选取的制约,并确立 $CL(e_y)$ 多路选取过程的终止条件,使得 p_j 不用指定 $CL(e_y)$ 大小,而是动态生成对网络状况敏感的因果控制信息,从而获得能够适应网络传输延迟变化的 $CL(e_y)$,既使 p_{des} 利用 $CL(e_y)$ 足以识别已传到的原因事件 e_x,又使 $CL(e_y)$ 大小与系统规模无关,减少冗余控制信息,降低传输和计算开销,有效维护 $e_x \xrightarrow{\Delta T} e_y$.

2.2 传输时间区间$[\Delta t_{\min}, \Delta t_{\max}]$

要想根据网络状况的变化动态选取出有效的 $CL(e_y)$,可以先根据文献[16-17]的网络坐标分布式算法探测网络传输双向延迟的大小,而在网络应用层的通信过程中,可以近似认为 p_i 与 p_j 的单向传输网络延迟时间 Δt_{ij} 为双向延迟值的一半.每个节点设置一个网络坐标向量,写入当前最新的网络坐标值,网络坐标调整后,会发送更新消息通知其他节点,但随着其值向稳态收敛,调整次数将逐渐减少,达到稳态后根据网络坐标计算的网络延迟能够较好反映当前的网络状态.

然而,通过上述计算得到的只是两点间的网络传输延迟,如果直接以它为基础研究因果一致性控制方法,p_j 最终只能计算出将 e_y 发送到某一接收节点的 $CL(e_y)$.若 p_j 需要将 e_y 发送到若干节点,则会导致 p_j 需要针对每一个接收节点,分别从因果关系中选取一次控制信息,然后再通过合并生成 $CL(e_y)$,包含大量重复计算会增大选取 $CL(e_y)$ 的计算开销,降低实时性.因此,必须先针对这种情况减少计算次数,为后续的 $CL(e_y)$ 选取加速.

如图 2 所示,p_j 与接收节点 p_i,p_k 的网络传输延迟存在差异,e_y 分别需要经历 Δt_1,Δt_2 的传输时间才能到达 p_i,p_k.然而进一步分析将会发现,虽然 e_y 需要经历不同的传输时间才能到达各接收节点,但它的传输时间范围却是确定的.即通过实时探测网络传输延迟,可以在 p_j 记录一个传输时间区间:$[\Delta t_{\min}, \Delta t_{\max}]$,其中 $\Delta t_{\min}$ 表示 e_y 传输到接收节点的最小传输时间,$\Delta t_{\max}$ 表示最大传输时间,则利用 $[\Delta t_{\min}, \Delta t_{\max}]$ 就能够得到 e_y 到达各接收节点的传输时间区间.而 p_j 发送 e_y 时间点为 $t^j_{e_y}$,因此 $[t^j_{e_y}+\Delta t_{\min}, t^j_{e_y}+\Delta t_{\max}]$ 表示了从 p_j 的时间系统来看 e_y 到达各接收节点的时间范围.同理,可以用 $[t^i_{e_x}+\Delta t_{x\min}, t^i_{e_x}+\Delta t_{x\max}]$ 代表 e_x 到达各接收节点的时间范围.这样,基于传输时间区间研究如何确定 $CL(e_y)$ 终止条件,动态选取恰当的 $CL(e_y)$,p_j 能够不必针对各接收节点分别进行计算,而是只通过一次选取就得到适合在 p_{des} 上维护 $e_x \xrightarrow{\Delta T} e_y$ 的 $CL(e_y)$,使 p_j 减少了计算次数,保证了实时性.

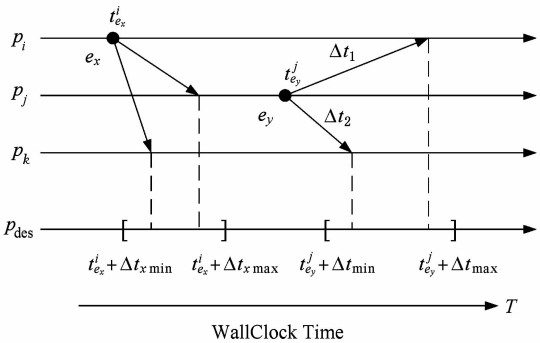


Fig. 2 The arriving time ranges of e_x, e_y on p_{des}.

图 2 e_x,e_y 到达 p_{des} 时间范围示意图

为了获取 $[\Delta t_{\min}, \Delta t_{\max}]$,$p_j$ 将在本地实时更新一个 n 维网络坐标向量,但 p_j 在每次发送事件消息前,并不需要遍历整个向量来计算 $[\Delta t_{\min}, \Delta t_{\max}]$,只有收到网络坐标更新消息且引起当前记录的 $[\Delta t_{\min}, \Delta t_{\max}]$ 发生变化才可能进行调整,维护开销比较小.

2.3 因果控制信息选取

为了选取有效的 $CL(e_y)$,在计算出事件的传输时间区间后,还要解决广域网时钟异步制约 $CL(e_y)$ 构建以及如何确定选取过程终止条件等问题.在本节我们针对上述问题进行研究,基于最小可计算时间点提出了时间点变换策略和终止条件的确立方法,并在此基础上给出了 $CL(e_y)$ 定义及其具体选取过程.

在图1所示例子中,p_{des}发现e_y生命周期结束时直接原因事件未及时传到,就要利用$CL(e_y)$从V_{des}的e_x中识别出可触发e_y处理的原因事件,然后逐次维护$e_x \xrightarrow{\Delta T} e_y$传递因果关系.然而,受网络状况变化的影响,在$p_{\mathrm{des}}$上可能出现$e_y$的原因事件连续迟到的情况,那么应如何动态构建$CL(e_y)$才能使其适应网络状况变化,既让$p_{\mathrm{des}}$从已传到$e_x$中能解析出触发$e_y$的原因事件,又不必传输过多的冗余信息呢?要解决这个问题,关键在于怎样确定$CL(e_y)$的选取方式及其选取过程的终止条件.从图1(a)看出,e_y的因果关系可以采用有向图描述,但要在整个图上直接选取出满足上述要求的$CL(e_y)$往往比较困难.通过分析可知,图1(a)中每条有向边表示了直接依赖关系,任意两事件间如果存在发生在先关系,则它们之间存在一条有向路径,该路径由一条或多条有向边组成.而互为并发关系的事件间不存在有向边.如图1(a)中$e_2 \parallel e_3$,$e_2 \parallel e_5$,则e_2与e_3,e_5之间没有有向边,也不存在因果先后关系.也就是说,p_j构建$CL(e_y)$时,因为不需要识别并发事件间的先后关系,可以考虑针对有向路径来分析$CL(e_y)$选取方式和终止条件,若p_{des}利用选出的$CL(e_y)$能够在各路径上识别出可作为触发条件的原因事件,就能传递各路径的因果关系,达到维护全图$e_x \xrightarrow{\Delta T} e_y$的目的.

下面针对图1(a)中的任意一条有向路径来分析如何选取$CL(e_y)$以及怎样确定终止条件.不失一般性,可以选择有向路径:$e_1 \rightarrow e_2 \rightarrow e_4 \rightarrow e_y$.由于是面向时钟异步的广域网分布式虚拟环境展开研究,为了更清楚地分析和解决问题,可将研究过程的侧重点分为两步:①假设时钟同步,确定$CL(e_y)$选取方式和终止条件;②解除时钟不同步制约,使①中结果在异步时钟环境内生效.

首先针对①进行研究.在$CL(e_y)$选取方式上,p_j从e_y的直接原因事件开始,沿着与有向路径相逆的方向,依次选取各原因事件标识信息写入$CL(e_y)$,直至满足终止条件.根据传输时间区间,p_j可以在自己时间系统内计算出e_y到达各接收节点p_{des}的时间范围是$[t^j_{e_y}+\Delta t_{\min}, t^j_{e_y}+\Delta t_{\max}]$,对在$p_j$已处理的原因事件$e_x$,可知其到达$p_{\mathrm{des}}$的时间范围是$[t^i_{e_x}+\Delta t_{x\min}, t^i_{e_x}+\Delta t_{x\max}]$,时间范围的覆盖区间会随网络状况的变化而动态调整,又因为假设各节点时间同步,所以可以比较不同节点的时间范围.在此基础上我们有以下结论.

定义6. 直接依赖关系可重构性.设$e_x \in R_i$,$e_y \in R_j$.若因果控制信息$CL(e_y)$标识的全体原因事件与e_y有形如$e_x \downarrow e_y$或$e_x \rightarrow e_1 \rightarrow \cdots \rightarrow e_m \rightarrow e_y$的因果关系,并满足:

$$\begin{cases} t^i_{e_x}+\Delta t_{x\max} \leqslant t^j_{e_y}+\Delta t_{\min},\ e_x \downarrow e_y; \\ (t^i_{e_x}+\Delta t_{x\max} \leqslant t^j_{e_y}+\Delta t_{\min}) \wedge \\ \quad (t^{i'}_{e_{x'}}+\Delta t_{x'\max} > t^j_{e_y}+\Delta t_{\min}), \\ \quad x' \in [1,m], e_{x'} \in R_{i'},\ \neg(e_x \downarrow e_y); \end{cases}$$

则称$CL(e_y)$具有直接依赖关系可重构性.

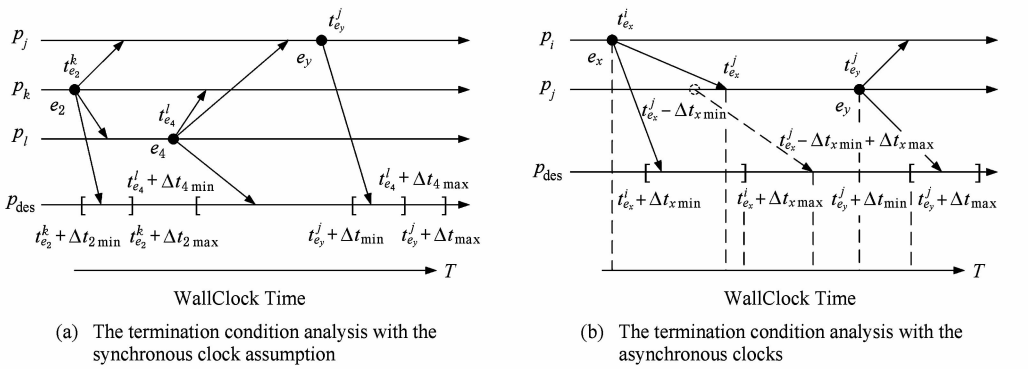
定理1. 如果p_j选取的因果控制信息$CL(e_y)$具有直接依赖关系可重构性,则p_{des}能利用$CL(e_y)$从V_{des}中识别出可触发e_y的原因事件,且不会选入冗余控制信息.

证明. 由于$t^{\mathrm{des}}_{e_x} \in [t^i_{e_x}+\Delta t_{x\min}, t^i_{e_x}+\Delta t_{x\max}]$,$t^{\mathrm{des}}_{e_y} \in [t^j_{e_y}+\Delta t_{\min}, t^j_{e_y}+\Delta t_{\max}]$,$e_y$生命周期结束时刻为$t^j_{e_y}+\Delta T$,而$\Delta T \geqslant \Delta t_{\min}$,则$e_y$生命周期结束时刻最早为$t^j_{e_y}+\Delta t_{\min}$.如果$CL(e_y)$能保证在$\Delta T=\Delta t_{\min}$,即生命周期结束时刻为$t^j_{e_y}+\Delta t_{\min}$情况下,从$V_{\mathrm{des}}$中识别出$t^{\mathrm{des}}_{e_x} \leqslant t^j_{e_y}+\Delta t_{\min}$的$e_x$触发$e_y$,则在$t^j_{e_y}+\Delta T$情况下必然也能触发$e_y$.因此,下面针对$\Delta T=\Delta t_{\min}$情况进行证明即可.由于$CL(e_y)$具有直接依赖关系可重构性,则如果$\neg(e_x \downarrow e_y)$,对$r(e_{x'}) \in CL(e_y)$,$e_{x'} \in R_{i'}$,$x' \in [1,m]$,$t^{\mathrm{des}}_{e_{x'}} \in [t^{i'}_{e_{x'}}+\Delta t_{x'\min}, t^{i'}_{e_{x'}}+\Delta t_{x'\max}]$,又$t^{i'}_{e_{x'}}+\Delta t_{x'\max} > t^j_{e_y}+\Delta t_{\min}$,由于$e_{x'} \rightarrow e_y$,所以$t^{i'}_{e_{x'}}+\Delta t_{x'\min} < t^j_{e_y}+\Delta t_{\min}$,因此,$[t^{i'}_{e_{x'}}+\Delta t_{x'\min}, t^{i'}_{e_{x'}}+\Delta t_{x'\max}] \cap [t^j_{e_y}+\Delta t_{\min}, t^j_{e_y}+\Delta t_{\max}] \neq \varnothing$.若在$p_{\mathrm{des}}$上满足$t^{\mathrm{des}}_{e_{x'}} \leqslant t^j_{e_y}+\Delta t_{\min}$,则在$(e_{x'} \downarrow e_y)$,或$x' < m$且$\forall x'' \in (x', m]$,$t^{\mathrm{des}}_{e_{x''}} > t^j_{e_y}+\Delta t_{\min}$时,$e_{x'}$是$V_{\mathrm{des}}$中触发$e_y$的原因事件;然而,若$p_{\mathrm{des}}$上$t^{\mathrm{des}}_{e_{x'}} > t^j_{e_y}+\Delta t_{\min}$,则$e_{x'}$不可能触发$e_y$,此时$r(e_{x'}) \in CL(e_y)$不足以让$p_{\mathrm{des}}$可重构$e_y$直接依赖关系.又因为$t^i_{e_x}+\Delta t_{x\max} \leqslant t^j_{e_y}+\Delta t_{\min}$,且$e_x \rightarrow e_y$,所以$[t^i_{e_x}+\Delta t_{x\min}, t^i_{e_x}+\Delta t_{x\max}] \cap [t^j_{e_y}+\Delta t_{\min}, t^j_{e_y}+\Delta t_{\max}] = \varnothing$,即对所有$p_{\mathrm{des}}$,$t^{\mathrm{des}}_{e_x} \leqslant t^j_{e_y}+\Delta t_{\min}$,则在$e_x \downarrow e_y$或$\forall x' \in [1,m]$,$t^{\mathrm{des}}_{e_{x'}} > t^j_{e_y}+\Delta t_{\min}$时,全部$p_{\mathrm{des}}$皆可识别$e_x$触发$e_y$.对具有直接依赖关系可重构性的$CL(e_y)$,若$t^{\mathrm{des}}_{e_{x'}} \leqslant t^j_{e_y}+\Delta t_{\min}$,则$e_{x'} \in V_{\mathrm{des}}$,如果$p_{\mathrm{des}}$要利用$CL(e_y)$识别$e_{x'}$来触发$e_y$,而若$r(e_{x'}) \notin CL(e_y)$,则$p_{\mathrm{des}}$无法从$V_{\mathrm{des}}$中识别$e_{x'}$,造成已传到原因事件的遗漏,因此,$CL(e_y)$没有选入冗余控制信息.

证毕.

由定理1可知,定义6的直接依赖关系可重构性表述了同步时钟下p_j在有向路径上为$CL(e_y)$依次选取原因事件标识信息的终止条件.当根据终止

条件找到第 1 个满足 $t^i_{e_x}+\Delta t_{x\max}\leqslant t^j_{e_y}+\Delta t_{\min}$ 的 e_x 后,p_{des}已能利用 $CL(e_y)$有效重建直接依赖关系,因此,无论是否还有满足该条件的原因事件,都不必再选取该路径的 $CL(e_y)$,否则会增加不必要的冗余传输开销. 如图 3(a)所示,$t^{\text{des}}_{e_4}\in[t^l_{e_4}+\Delta t_{4\min},t^l_{e_4}+\Delta t_{4\max}]$,$t^l_{e_4}+\Delta t_{4\max}\geqslant t^j_{e_y}+\Delta t_{\min}$,由于 $e_4\rightarrow e_y$,$t^l_{e_4}+\Delta t_{4\min}<t^j_{e_y}+\Delta t_{\min}$,所以$[t^l_{e_4}+\Delta t_{4\min},t^l_{e_4}+\Delta t_{4\max}]\cap[t^j_{e_y}+\Delta t_{\min},t^j_{e_y}+\Delta t_{\max}]\neq\varnothing$. 若 $t^{\text{des}}_{e_4}\leqslant t^{\text{des}}_{e_y}+\Delta T$,则 e_4 就是触发 e_y 被处理的原因事件,p_j 将 $r(e_4)\in CL(e_y)$即可;而若 $t^{\text{des}}_{e_4}>t^j_{e_y}+\Delta T$,则因只有 $r(e_4)\in CL(e_y)$,p_{des}将不能触发处理 e_y. 而 $t^k_{e_2}+\Delta t_{2\min}<t^k_{e_2}+\Delta t_{2\max}<t^j_{e_y}+\Delta t_{\min}$,则$[t^k_{e_2}+\Delta t_{2\min},t^k_{e_2}+\Delta t_{2\max}]\cap[t^j_{e_y}+\Delta t_{\min},t^j_{e_y}+\Delta t_{\max}]=\varnothing$,即在各 p_{des}上,$t^{\text{des}}_{e_2}\leqslant t^{\text{des}}_{e_y}+\Delta T$,若 p_j令 $r(e_2)\in CL(e_y)$,则即使 $t^{\text{des}}_{e_4}>t^j_{e_y}+\Delta T$ 时,仍可使 e_2 作为触发 e_y 的原因事件,从而传递因果关系. 因此,在选出 $r(e_2)$,$r(e_4)\in CL(e_y)$ 后,p_j 则不需要考虑再选取其他原因事件标识.



(a) The termination condition analysis with the synchronous clock assumption

(b) The termination condition analysis with the asynchronous clocks

Fig. 3 The analysis of the termination condition of $CL(e_y)$ selection.

图 3 $CL(e_y)$终止条件分析过程示意图

然而,定理 1 只能在时钟同步的假设下起作用,还必须进一步解决前面②的解除时钟不同步制约中问题. 在广域网运行的分布式大规模虚拟环境中,所有计算节点的时钟要完全达到精确同步开销较大,将制约使用定理 1 判断何时终止 $CL(e_y)$选取的有效性. 由于时间间隔是相对的时间差值,而各节点时间步进速率差异非常小,所以仿真过程中用于计算的时间间隔在各节点都有意义. 因此,如果能够找到间接的时间值使不同节点时间可以比较先后,则在时钟不同步情况下确定 $CL(e_y)$终止条件将成为可能. 为此本文提出了时间点变换策略. 如图 3(b)所示,因为 $e_x\rightarrow e_y$,所以若各节点有统一的墙钟时间,则墙钟时间对应时刻上 $t^i_{e_x}$ 应先于 $t^j_{e_y}$ 发生,但由于不存在统一的授时,p_j 与 p_i 时钟异步,p_j 收到 e_x 后不能直接判断 $t^i_{e_x}+\Delta t_{x\max}$ 与 $t^j_{e_y}+\Delta t_{\min}$ 的先后关系,无法用定理 1 选取 $CL(e_y)$. 但 p_j 在自己的时间系统内能识别接收 e_x 的时间点,记为 $t^j_{e_x}$. 虽然不存在统一授时的墙钟时间,但真实时间流逝是客观存在的,可以将其认为是客观墙钟时间,基于先发后收的物理常识,$t^j_{e_x}$ 对应的客观墙钟时刻应晚于 $t^i_{e_x}$ 相应客观时刻发生. 因此,可以定义客观墙钟时间下的异步时钟比较关系.

定义 7. 异步时钟比较关系. 设 t^i 为 p_i 时间系统内的时间点,t^j 为 p_j 时间系统内的时间点,如果在客观流逝的时间上能够判断 t^i 早于 t^j 发生,则记为 $t^i<t^j$;如果在客观墙钟时间上能够判断 t^i 晚于 t^j 发生,则记为 $t^i>t^j$.

为了在异步时钟下使 $CL(e_y)$尽可能减少选入冗余信息,需要寻找合适的变换时间点.

定义 8. 最小可计算时间点. 设异步时钟条件下 $e_x\in R_i$,$e_y\in R_j$,$e_x\rightarrow e_y$,则 p_j 时间系统内可计算的、对应在客观墙钟时间上晚于 $t^i_{e_x}$ 相应客观时刻的最小时间点,称为 p_j 上与 $t^i_{e_x}$ 相关的最小可计算时间点.

在图 3(b)中,利用已有信息,p_j 获知 e_x 发送后最少也需经历间隔 $\Delta t_{x\min}$ 才能传到,则可以计算出 $t^j_{e_x}-\Delta t_{x\min}$ 对应的客观墙钟时刻应等于或晚于 $t^i_{e_x}$ 相应的客观时刻,而 $t^j_{e_x}-\Delta t_{x\max}$ 则等于或早于该时刻,所以 $t^j_{e_x}-\Delta t_{x\min}$ 是 p_j 在当前情况下所能识别的、与 $t^i_{e_x}$ 相关的最小可计算时间点. 利用最小可计算时间点能够给出异步时钟下的 $CL(e_y)$的直接依赖关系可重构性.

定义 9. 异步时钟直接依赖关系可重构性. 设 $e_x\in R_i$,$e_y\in R_j$. 若因果控制信息 $CL(e_y)$标识的全体原因事件与 e_y 有形如 $e_x\downarrow e_y$ 或 $e_x\rightarrow e_1\rightarrow\cdots\rightarrow e_m\rightarrow e_y$

的因果关系,并满足:

$$\begin{cases} t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max} \leqslant t^j_{e_y} + \Delta t_{\min},\ e_x \downarrow e_y; \\ (t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max} \leqslant t^j_{e_y} + \Delta t_{\min}) \wedge \\ \quad (t^j_{e_{x'}} - \Delta t_{x'\min} + \Delta t_{x'\max} > t^j_{e_y} + \Delta t_{\min}), \\ \quad x' \in [1,m], e_{x'} \in R_{i'},\ \neg(e_x \downarrow e_y); \end{cases}$$

则称 $CL(e_y)$具有异步时钟直接依赖关系可重构性.

定理 2. 如果 p_j 选取的因果控制信息 $CL(e_y)$具有异步时钟直接依赖关系可重构性,则可终止 $CL(e_y)$选取.

证明. $\forall r(e_x) \in CL(e_y)$,因为$(t^j_{e_x} - \Delta t_{x\min}) > t^i_{e_x}$,所以$(t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max}) > (t^i_{e_x} + \Delta t_{x\max})$,即在客观墙钟时间到达与 $t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max}$相应的时刻时,在各 p_{des}上 $e_x \in V_{\text{des}}$成立. 又因为 $t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max} \leqslant t^j_{e_y} + \Delta t_{\min}$,则 $t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max} < t^{\text{des}}_{e_y}$,因此,$(t^i_{e_x} + \Delta t_{x\max}) < t^{\text{des}}_{e_y}$. 如果 $e_x \downarrow e_y$,或 $\neg(e_x \downarrow e_y)$且 $\forall r(e_{x'}) \in CL(e_y)$,$e_{x'} \in R_{i'}$,$x' \in [1,m]$,$e_{x'}$均在 e_y 生命周期结束后到达,则 p_{des}可利用 $CL(e_y)$识别 e_x 触发 e_y. 又因为 $CL(e_y)$从直接原因开始依次选取标识,所以当有 $e_{x'}$在 e_y 生命周期结束前传到,p_{des}能够通过 $CL(e_y)$识别出 $e_{x'}$,不会遗漏因果关系. 证毕.

通过上述分析得出的 $CL(e_y)$选取方式及定理 2 终止条件,可在异步时钟情况下被应用于图 1(a)各有向路径,从而能获得能够维护全图 $e_x \xrightarrow{\Delta T} e_y$ 的因果控制信息,其定义如下:

定义 10. 因果控制信息. 设 $e_x \in R_i$,$e_y \in R_j$,p_{des}是 e_x,e_y 的共同接收节点且 $e_x \rightarrow e_y$. p_j 发送 e_y 前,为 e_y 选取因果控制信息 $CL(e_y) = \{element_{e_x} \mid element_{e_x} = (r(e_x), t^j_{e_x}, [\Delta t_{x\min}, \Delta t_{x\max}])\}$,使 p_{des}接收 e_y 后,能利用 $CL(e_y)$有效维护 $e_x \xrightarrow{\Delta T} e_y$. 其中,$r(e_x)$能使 p_{des}识别已传到原因事件,$t^j_{e_x}$与$[\Delta t_{x\min}, \Delta t_{x\max}]$是 LCO 控制方法在各节点能够持续运行所需的数据变量,$element_{e_x}$代表了 e_x 为因果一致性控制提供的相关状态信息. p_j将从 e_y 直接原因开始,依次添加 $element_{e_x}$到 $CL(e_y)$中,直至满足定理 2 终止条件. 而选取出的 $CL(e_y)$内容会随网络状况变化而动态调整,使 $CL(e_y)$能适应网络传输延迟变化,使 p_{des}收到 e_y 后,可以通过 $CL(e_y)$在各路径上解析出触发 e_y 被处理的原因事件,传递因果关系,维护 $e_x \xrightarrow{\Delta T} e_y$. 同时 $CL(e_y)$大小与系统规模无关,可以减少冗余控制信息,降低传输和计算开销. 在构建 $CL(e_y)$的过程中,因为是基于传输时间区间来进行比较,所以可一次选取出能够在各 p_{des}上维护 $e_x \xrightarrow{\Delta T} e_y$ 的 $CL(e_y)$,减少计算次数,提高了实时性.

下面将给出 $CL(e_y)$的具体选取过程,在此之前先说明需要用到的各个变量.

$VT(p_j)$:p_j 的逻辑时间向量,记录了它当前所知的各个节点最新的逻辑时间推进情况,其中 $VT(p_j)[j]$表示 p_j 的逻辑时间.

$CG(p_j)$:缓存在 p_j 本地的邻接表,用于记录、更新已收到和发送事件间的因果关系图,为选取 $CL(e_y)$提供原因事件标识及其相应状态信息. 表中各元素为四元组$(r(e_x), t^j_{e_x}, [\Delta t_{x\min}, \Delta t_{x\max}], p_cs[num])$,其中 $p_cs[num]$是指针数组,分别指向 e_x 的直接原因事件在 $CG(p_j)$中的相应元素,num 表示连入 e_x 的有向路径数. 为了避免 $CG(p_j)$逐渐增大,p_j 会定期删除 $CG(p_j)$无效冗余元素. 假设 p_j 当前的时间为 t^j,则 $t^j < t^j + \Delta t_{\min}$,若在 $CG(p_j)$中检测到某个 e_x 的对应元素满足 $t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max} < t^j$,则 $t^j_{e_x} - \Delta t_{x\min} + \Delta t_{x\max} \leqslant t^j + \Delta t_{\min}$成立,那么它自己本身有可能满足定理 2 终止条件,但其原因事件已无法满足,因此,可定期将这些元素从 $CG(p_j)$中删除.

$CL(e_y)$:p_j 用于存储待发送事件 e_y 的因果控制信息的向量,它将被放置到包含 e_y 的消息 M 中,一同向外发送. 因为程序实现需要添加变量记录 $CL(e_y)$各 $element_{e_x}$之间的先后关系,所以 $CL(e_y)$各 $element_{e_x}$也是四元组$(r(e_x), t^j_{e_x}, [\Delta t_{x\min}, \Delta t_{x\max}], p_cs[num])$,前 4 个变量可相应从 $CQ(s_j)$中获取,$p_cs[num]$记录的是 e_x 各原因事件元素在 $CL(e_y)$中的位置索引.

CD_M:在消息 M 中为 p_{des}查询 $CL(e_y)$提供起始位置索引的向量,各分量由二元组$(r(e_x), d_cs)$构成,其中 d_cs 是 e_y 的直接原因事件元素在 $CL(e_y)$的位置索引.

$CN(p_j)$:在 p_j 本地为查询 $CG(p_j)$提供起始指针的向量,各分量由二元组$(r(e_x), f_cs)$构成,其中 f_cs 是指向当前 $CG(p_j)$中直接原因事件元素的指针.

$CL(e_y)$的具体选取过程如下:

```
Procedure SelectCL(svi, sn, cg)   /* p_j 为 e_y 选取 CL(e_y) */
{/* svi 为上层调用的元素在 CL(e_y)中的位置索引值,sn 为该元素 num 编号,cg 为 CG(p_j)中对应元素的 p_cs[sn]指针 */
  if (!FindInCG(cg)) return false;   /* 若在 CG(s_j)中找不到 cg 指向的元素,表示到
```

```
达路径尾端或产生异常 */
vi = FindInCL(cg.r(e_x)); /* 若能找到,通过r(e_x)查询该元素是否已加入CL(e_y) */
if(vi){CL(e_y)[svi].p_cs[sn]=vi;return ture;} /* 若已加入,向上层调用返回此元素在CL(e_y)中的位置索引 */
vi=CL(e_y).add(*cg); /* 若未加入,将其添加到CL(e_y)中 */
if(svi==null && sn==null) /* 若添加的元素对应是e_y的直接原因 */
  CD_M.add(CL(e_y)[vi].r(e_x),vi); /* 将该元素在CL(e_y)中的位置索引记录在CD_M中 */
else /* 若添加的元素对应是e_y的间接原因 */
  CL(e_y)[svi].p_cs[sn]=vi; /* 将其位置索引返回给上层调用元素 */
if(CL(e_y)[vi].(t^j_{e_x} − Δt_{x min} + Δt_{x max}) ≤ (t^j_{e_y} + Δt_min)) /* 若此次加入CL(e_y)的元素已满足定理2终止条件 */
  for(iter=0;iter<num;iter++) /* 在其所在的CG(s_j)有向路径上停止选取其他r(e_x) */
    CL(e_y)[vi].p_cs[iter]=null;
else /* 若仍不能满足定理2终止条件 */
  for(iter=0;iter<num;iter++) /* 继续为CL(e_y)依次递归选取其他r(e_x) */
    if(! SelectCL(vi,iter,CG(p_j)[cg].p_cs[iter]))
      CL(e_y)[vi].p_cs[iter]=null;
return true;
}
```

2.4 消息发送算法

选取出 $CL(e_y)$ 及提供起始索引 CD_M 后,p_j 将更新本地逻辑时间,并将 e_y,CD_M,$CL(e_y)$,ΔT 和逻辑时间等数据封装在消息 M 中向 p_{des} 发送,而发送完 M,e_y 将是当前 p_j 最新发生的事件,需要将包含 $r(e_y)$ 的四元组 $(r(e_y), t^j_{e_y}, [\Delta t_{\min}, \Delta t_{\max}], p\_cs[num])$ 加入到 $CG(p_j)$ 中,其中 $r(e_y)=(j, VT(p_j)[j])$.

为了使 p_{des} 能够有效在 e_y 生命周期内完成因果一致性控制,消息 M 的格式被设置为如图 4 所示:

e_y	j	$VT(p_j)[j]$	$t^j_{e_y}$	$[\Delta t_{\min}, \Delta t_{\max}]$	ΔT	CD_M	$CL(e_y)$

Fig. 4 The format of message M.

图 4 消息 M 格式示意图

下面给出 e_y 的消息发送算法.

算法 1. 消息发送算法.

① $VT(p_j)[j]=VT(p_j)[j]+1$; /* 更新 p_j 逻辑时间 */

② $t^j_{e_y}=timeGetTime()$; /* 获取发送 e_y 的 p_j 本地时间戳 */

③ for($iter=0$;$iter<CN(p_j).size()$;$iter++$) /* 从起始向量开始查询 $CG(p_j)$,计算 $CL(e_y)$ 和 CD_M */

④ $SelectCL(\text{null}, \text{null}, CN(p_j)[iter].f\_cs)$;

⑤ $M \leftarrow (e_y, j, VT(p_j)[j], t^j_{e_y}, [\Delta t_{\min}, \Delta t_{\max}], \Delta T, CD_M, CL(e_y))$; /* 获得包含 e_y 和控制信息 $CL(e_y)$ 等数据的 M */

⑥ $SendMessage(M)$; /* 发送消息 M */

⑦ $ptr=CG(p_j).add(r(e_y), t^j_{e_y}, [\Delta t_{\min}, \Delta t_{\max}], p\_cs[CN(p_j).size()])$; /* 在 $CG(p_j)$ 加入 e_y 的四元组并返回其指针 */

⑧ for($iter=0$;$iter<CN(p_j).size()$;$iter++$) /* 使 e_y 对应的 $p\_cs[num]$ 分别指向 $CG(p_j)$ 中原来的起始原因事件元素 */

⑨ $ptr\rightarrow p\_cs[iter]=CN(p_j)[iter].f\_cs$; /* 起始原因事件元素 */

⑩ $CN(p_j).clear$; /* 清空 $CN(p_j)$ 中原来的起始指针 */

⑪ $CN(p_j).add(r(e_y), ptr)$; /* 加入 ptr,作为 $CN(p_j)$ 新的指向 $CG(p_j)$ 的起始指针 */

⑫ $CL(e_y).clear()$; /* 清空 $CL(e_y)$ 向量 */

⑬ $CD_M.clear()$; /* 清空 CD_M */

⑭ $exit()$; /* 算法结束退出 */

2.5 消息接收算法

p_{des} 收到包含 e_y 的消息 M 后,需要针对 M 进行判断,接收算法对不同的判断结果采取了不同的处理策略.而在时钟异步情况下,e_y 在 p_{des} 的最小可计算时间点为 $t^{\text{des}}_{e_y}-\Delta t_{\min}$,它是当前信息下所能获取的晚于且最逼近 $t^j_{e_y}$ 对应客观墙钟时刻的时间点,因此利用它可近似计算出 p_{des} 上 e_y 生命周期的结束时刻,即 $t^{\text{des}}_{e_y}-\Delta t_{\min}+\Delta T$. 如果 e_y 生命周期结束时发生了原因事件失效的情况,p_{des} 依旧可以利用 $CL(e_y)$

找出已传到的 e_x,维护 $e_x \xrightarrow{\Delta T} e_y$,并通过构建空消息和计算虚拟接收时间点,保持本地因果关系的完整性.

2.5.1 消息接收

M 到达 p_{des} 时存在几种可能情况,若 M 为未及时传到的失效消息,则其结果事件消息已经被处理,M 被 p_{des} 抛弃;若 M 有效,则 p_{des} 判断 e_y 的各直接原因事件是否已被执行,如果都执行过 M 可尽快处理,不需要等到生命周期 ΔT 结束;否则 p_{des} 记录 M 的接收时间,并一起缓冲至消息队列 $MQ(p_{\text{des}})$.

$MQ(p_{\text{des}})$ 为 p_{des} 的消息缓存队列,存放直接原因事件未被处理而生命周期又没有结束的事件消息. p_{des} 将实时扫描 MQ,若发现某消息直接原因事件已全部执行完,或已到达生命周期结束时刻,或被后续结果事件消息要求提前执行,则 p_{des} 将消息及其接收时间从 MQ 中提取出来进行处理.

算法 2. 消息接收算法.

① $t_{e_y}^{\text{des}} = timeGetTime()$; /* 获取 p_{des} 接收 e_y 的本地时间 */

② if($VT(p_j)[j] \leqslant VT(p_{\text{des}})[j]$) /* 若 M 已失效 */

③ {$AbandonMessage(M)$; $exit()$;} /* 则丢弃消息,并退出接收算法 */

④ else /* 若 M 有效 */

{

⑤ for($iter=0$; $iter<CD_M.size()$; $iter++$) /* 则判断 M 是否能够尽快处理 */

⑥ if($CD_M[iter].r(e_x).a > VT(p_{\text{des}})[CD_M[iter].r(e_x).i]$) /* 若有直接原因事件未执行 */

⑦ {$BufferMessage(M, t_{e_y}^{\text{des}})$; $exit()$;} /* 将 M 与接收时间一起缓冲至 MQ,退出接收算法 */

⑧ $HandleMessage(M, t_{e_y}^{\text{des}})$; /* 若直接原因事件都已在 p_{des} 执行完,则尽快处理 M */

⑨ $exit()$;} /* 退出接收算法 */

2.5.2 使用因果控制信息处理消息

P_{des} 处理消息 M 时,通过递归调用消息处理函数 $HandleMessage(M, t_{e_y}^{\text{des}})$ 解析因果控制信息 $CL(e_y)$,能够利用 $r(e_x)$ 逐次查询到可触发 e_y 被处理的原因事件以及其他已传到未处理的 e_x,并使它们按照因果关系的先后顺序正确执行,从而维护 $e_x \xrightarrow{\Delta T} e_y$,传递因果关系.

如果遇到有原因事件失效的情况,P_{des} 也仍旧可以利用因果控制信息为失效事件构建空消息,从而使递归调用 $HandleMessage(M, t_{e_y}^{\text{des}})$ 处理原因事件消息能够持续进行. 然而,空消息代替的失效原因事件并未真正被 p_{des} 及时接收和执行,为了保证记录 p_{des} 本地因果关系的 $CG(p_{\text{des}})$ 的完整性,使 p_{des} 以后发送消息时能有效利用定理 2 终止条件选出因果控制信息,必须为空消息对应加入 $CG(p_{\text{des}})$ 的元素添加一个接收时间点,称为虚拟接收时间点. 若空消息是通过 $CL(e_y)$ 构建,则失效原因事件 e_x 在 p_{des} 虚拟接收时间点应根据 e_y 提供的相关信息计算,其计算原则是使 p_{des} 利用虚拟接收时间点能够计算出以 e_y 相关信息为基础的最小可计算时间点. 如图 5 所示,$e_x \rightarrow e_y$,在 p_{des} 处理 e_y 过程中,发现 e_x 未能及时传到 p_{des},$t_{e_x}^j$ 是 e_x 在 e_y 的真实接收时间点,为了得到 e_x 在 p_{des} 的虚拟接收时间点,可参照 e_x 在 p_j 最小可计算时间点的计算过程,相应计算出 e_x 在 p_{des} 的最小可计算时间点为 $t_{e_y}^{\text{des}} - \Delta t_{\min} - (t_{e_y}^j - t_{e_x}^j) - \Delta t_{x\min}$,则 e_x 在 p_{des} 的虚拟接收时间点为 $t_{e_y}^{\text{des}} - \Delta t_{\min} - (t_{e_y}^j - t_{e_x}^j)$,将其记录在本地因果关系 $CG(p_{\text{des}})$ 的对应元素中,保证了 $CG(p_{\text{des}})$ 的完整性,同时也满足了 p_{des} 运用定理 2 的基本条件.

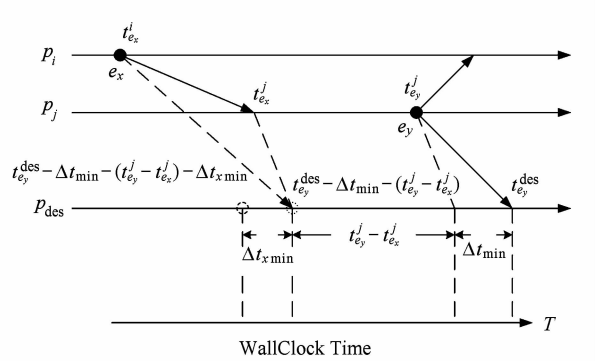


Fig. 5 The computation of virtual receiving time of e_x on p_{des}.

图 5 e_x 在 p_{des} 虚拟接收时间点计算示意图

消息处理函数 $HandleMessage(M, t_{e_y}^{\text{des}})$ 具体过程如下:

Procedure $HandleMessage(M, t_{e_y}^{\text{des}})$ /* p_{des} 处理 M,接收 M 时间为 $t_{e_y}^{\text{des}}$ */

{$fp = CG(p_{\text{des}}).add(r(e_y), t_{e_y}^{\text{des}}, [\Delta t_{\min}, \Delta t_{\max}], p\_cs[CD_M.size()])$; /* 将 e_y 的四元组加入 $CG(p_{\text{des}})$ 并返回其指针 */

for($iter=0$; $iter<CD_M.size()$; $iter++$)

/* 判断 e_y 的各直接原因事件是否已执行 */

{if($CD_M[iter].r(e_x).a \leqslant VT(p_{des})[CD_M[iter].r(e_x).i]$) /* 若直接原因已执行 */
$fp \rightarrow p\_cs[iter] = FindInCG(CD_M[iter].r(e_x))$; /* 返回其四元组在 $CG(p_{des})$ 的指针 */
else /* 若未执行 */
{$M' = FindInMQ(CD_M[iter].r(e_x))$;
/* 查询直接原因事件消息 M' 是否在 MQ 中 */
if(M') $fp \rightarrow p\_cs[iter] = HandleMessage(M', t_{e_x}^{des})$; /* 若在 MQ 中,递归处理 M' */
else /* 若不在,直接原因事件失效,构建空消息 M'' */
{$tmp = CL(e_y)[CD_M[iter].d\_cs]$; /* 获取直接原因四元组在 $CL(e_y)$ 中位置 */
for($iter\_1 = 0$; $iter\_1 < tmp.p\_cs[num]$; $iter\_1$++) /* 进一步利用该四元组自己的各直接原因 */
$CD_{M'}.add(tmp.p\_cs[iter\_1].r(e_x), tmp.p\_cs[iter\_1])$; /* 建立 $CD_{M'}$ */
$M'' \leftarrow$(null, $tmp.r(e_x).i$, $tmp.r(e_x).a$, null, $tmp.[\Delta t_{x\,min}, \Delta t_{x\,max}]$, null, $CD_{M'}$, $CL(e_y)$); /* 空消息 M'' */
$t_{virtual} = t_{e_y}^{des} - \Delta t_{min} - (t_{e_y}^{j} - tmp.t_{e_x}^{j})$;
/* 计算失效事件在 p_{des} 的虚拟接收时间点 */
$fp \rightarrow p\_cs[iter] = HandleMessage(M'', t_{virtual})$;} /* 递归处理 M'',持续查找已传到原因事件 */
}
for($iter\_2 = 0$; $iter\_2 < CN(p_{des}).size()$; $iter\_2$++) /* 若 $CN(p_{des})$ 含有指向直接原因事件四元组 */
if($CD_M[iter].r(e_x) == CN(p_{des})[iter\_2].r(e_x)$) $CN(p_{des}).remove(iter\_2)$;
/* 删除 $CN(p_{des})$ 相应元素 */
}
$CN(p_{des}).add(r(e_y), fp)$;
/* 在 $CN(p_{des})$ 指向 e_y 四元组的指针 */
if($VT(p_j)[j] > VT(p_{des})[j]$)
$VT(p_{des})[j] = VT(p_j)[j]$;
/* 更新 $VT(p_{des})$ 逻辑时钟 */
$DeliveryEvent(e_y)$; /* 处理事件 e_y */
return fp; /* 返回指向 e_y 四元组的指针 */
}

3 实验结果与分析

为了验证本文所提出的带生命周期约束因果一致性控制方法的有效性,我们在国防科大并行与分布处理国防科技重点实验室集群中心构建了分布式时空因果一致性模拟验证环境,底层以北航 BH-RTI[18-19]作为运行时支撑结构,中间层设计了网络坐标计算模块和因果一致性控制算法,上层以模拟大规模红蓝方飞机实体空中对战作为分布式仿真测试实例,验证环境体系结构如图 6 所示. 为了模拟广域网延迟情况,验证环境基于 Spirent ConNIE 硬件仿真仪实现了实时的数据延迟损伤处理. 事件和因果控制信息封装在消息中一同对外发送,消息序设置为接收序(receive order, RO)类型,接收实体通过因果控制信息进行因果事件一致性维护. 各消息生命周期由相应发送实体计算和赋值.

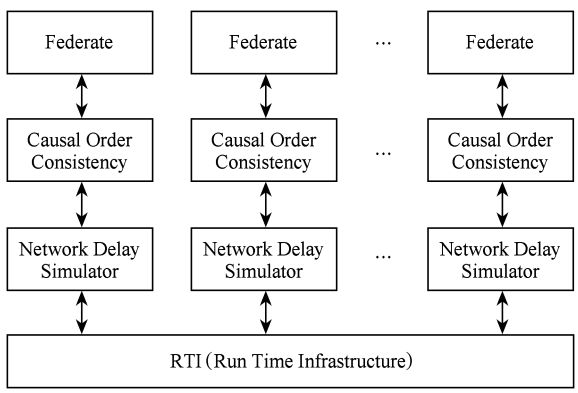


Fig. 6 The architecture of the causal order consistency simulation and verification environment.

图 6 因果一致性模拟验证环境体系结构图

在实验过程中,我们通过与其他因果一致性控制方法的比较来验证 LCO 的效果. 因为 LCO 是针对异步时钟场景提出的因果一致性控制方法,所以实验环境设置仿真时钟不同步,而以文献[3]和文献[4]为代表的带生命周期约束因果一致性控制方法需要在时钟精确同步条件下才能运行,不能有效地参与比较,因此在实验中 LCO 将与经典因果一致性控制方法进行对比. 对基于向量时间的一致性控制方法采用文献[7]给出的代表性方法 Vector Time;对基于直接原因的一致性控制方法采用文献[11]的

代表性方法 IDR(immediate dependency relation). 在多次实验中,空战演练飞机实体规模依次设定为 3 600 架、5 400 架、7 200 架、9 000 架和10 800 架,均匀分布在集群节点上,双方飞机实体在同一战场环境中对战,运行界面如图 7 所示:

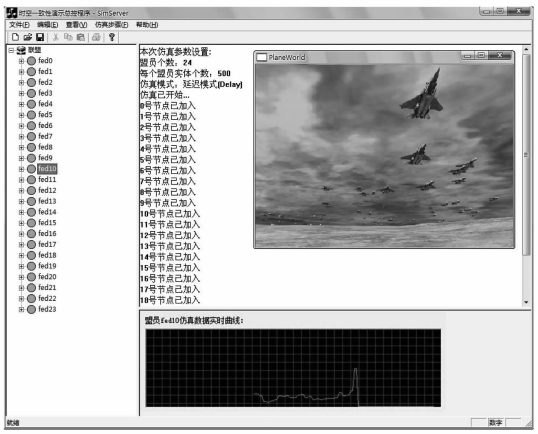


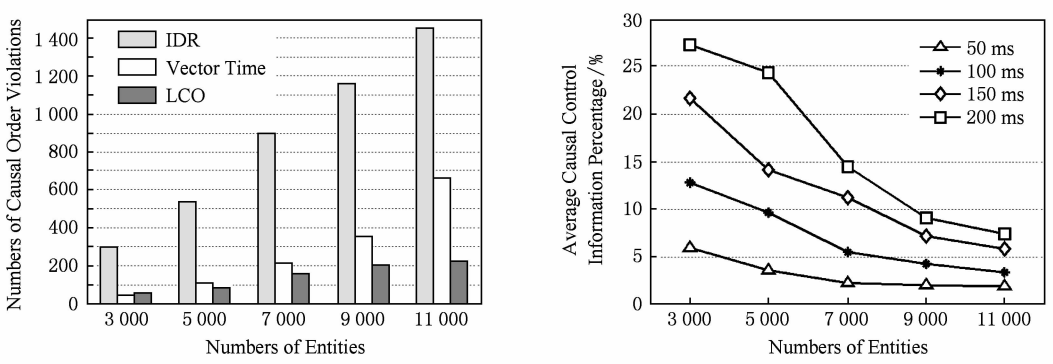
Fig. 7 The GUI of the causal order consistency simulation and verification environment.

图 7 因果一致性模拟验证环境运行界面

由于因果一致性维护的正确性可从因果违背次数进行分析,因此我们分别测试和统计了 3 种方法在各仿真规模下的因果不一致发生次数,并进行了比较;而因果一致性控制方法的消息传输和计算开销可从消息携带的因果控制信息平均大小来衡量,为此我们针对不同的网络条件在各实体规模下进行了实验,并分析了因果控制信息平均大小与实体规模的变化关系. 实验结果如图 8 所示.

在图 8(a)中,因为 IDR 的消息因果控制信息只包含了直接原因信息,一旦直接原因事件没有在结果事件生命周期结束前及时传到,就无法查询已到达的间接原因,使原因事件不能在结果事件前得到处理,容易发生因果违背情况,因此,在各种规模下 IDR 的因果不一致次数都是最多的,3 000 规模时接近 300,而 11 000 规模则超过了 1 400. Vector Time 完整记录了各个实体的逻辑时钟,传输和计算规模为 $O(n)$,事件消息生命周期结束时,由于其控制和计算开销过大,不能及时找出所有已传到原因事件,产生因果不一致现象,但规模不大时影响较小,3 000 规模时低于 100,而随着规模增大,违背次数增加较快,9 000 规模时约增加到 7 000 规模的 166%,11 000 规模较 9 000 规模约增加到 187%. LCO 正确性与网络状况相关,与实体规模无关,在 3 000 规模时,不一致次数略高于 Vector Time,但在 3 000 规模以上违背程度和次数都低于 Vector Time,尤其在实体规模增加到 11 000 时,不一致次数控制在 200 左右,约为同等规模 Vector Time 违背次数的 30%、IDR 的 15%,良好地提高了因果一致性维护的正确性.

图 8(b)给出了多种网络条件下 LCO 因果控制信息平均大小与 Vector Time 因果控制信息的百分比,及其随实体规模增加的变化关系. Vector Time 的传输和计算规模为 $O(n)$,与实体规模紧密耦合,随规模增加而快速增大. 从图中可知,在网络平均延迟为 50 ms 和实体规模为 3 000 时,LCO 因果控制信息平均大小约为 Vector Time 的 6%,在网络平均延迟增大后,不能及时传到的原因事件可能增多,为了仍旧能够有效传递因果关系,LCO 因果控制信息相应增大,但平均延迟为 200 ms 时也只增加到 Vector Time 因果控制信息大小的 27%左右. 随着实体规模增加,Vector Time 控制信息逐步增大,但 LCO 因果控制信息大小与实体规模无关,所以在各种网络条件下 LCO 因果控制信息平均大小百分比都随规模增大而呈下降趋势,在 3 000 规模时,平均网络延迟 50 ms,100 ms,150 ms,200 ms 条件下的



(a) The number of causal order violations

(b) The average causal control information percentage

Fig. 8 The evaluation of experimental results of causal order consistency control methods

图 8 因果一致性控制方法实验结果

LCO 因果控制信息平均大小百分比分别约为 6%,13%,22%,27%,当规模增加到 11 000 时,各网络条件下的相应 LCO 因果控制信息平均大小百分比已逐渐降低至 2%,3%,6%,7%左右,相比 Vector Time 有效降低了传输和计算开销,在满足分布式大规模虚拟环境实时性要求的同时,较好地提高了环境的可扩展性.

4 结束语

在广域网运行的分布式大规模虚拟环境中,维护事件间因果一致性要满足实时性要求必须在事件生命周期内完成.而网络传输的高延迟和动态性会导致部分原因事件无法在结果事件的生命周期内及时传到,如果不有效处理将会破坏已到达事件间的因果关系.因此,本文提出一种新的生命周期约束下的因果一致性控制方法 LCO,采用时间点变换策略突破时钟异步对因果控制信息选取的制约,基于网络传输时间区间确立多路选取因果控制信息的终止条件,使因果控制信息内容随网络状况动态调节,进而在已传到的事件中识别出可重构直接依赖关系的非直接原因事件,使因果一致性维护不受未达到事件的影响.实验结果表明,与以往控制方法相比,LCO 既可以减少因果不一致现象,有效传递事件因果关系,又使因果控制信息量与环境规模无关,降低传输和计算开销,从而能够实时维护生命周期约束下的因果一致性.

参 考 文 献

[1] Fujimoto R M. Parallel and Distributed Simulation Systems [M]. New York: Wiley Interscience, 2000: 195-275

[2] Strassburger S, Schulze T, Fujimoto R M. Survey on future trends in distributed simulation and distributed virtual environments: Results of a peer study [C] //Proc of the 2008 Winter Simulation Conf. Piscataway, NJ: IEEE, 2008: 777-785

[3] Baldoni R, Prakash R, Raynal M, et al. Efficient Δ-causal broadcasting [J]. Journal of Computer System Science and Engineering, 1998, 13(5): 263-269

[4] Rodrigues L, Baldoni R, Anceaume E, et al. Deadline-constrained causal order [C] //Proc of the 3d IEEE Int Symp on Object-oriented Real-time distributed Computing. Piscataway, NJ: IEEE, 2000

[5] Declan D, Tomas W, Seamus M. On consistency and network latency in distributed interactive applications: A survey—part I [J]. Presence: Teleoperators and Virtual Environments, 2006, 15(2): 218-234

[6] Schiper A, Eggli J, Sandoz A. A new algorithm to implement causal ordering [C] //Proc of the 3rd Int Workshop on Distributed Algorithms. Berlin: Springer, 1989: 219-232

[7] Schwarz R, Mattern F. Detecting causal relationships in distributed computations: in search of the holy grail [J]. Distributed Computing, 1994, 7(3): 149-174

[8] Cai W T, Turner S J, Lee B S, et al. An alternative time management mechanism for distributed simulations [J]. ACM Trans on Modeling and Computing Simulation, 2005, 15(2): 109-137

[9] Cai W T, Lee B S, Zhou J L. Causal order delivery in a multicast environment: an improved algorithm [J]. Journal of Parallel and Distributed Computing, 2002, 62(1): 111-131

[10] Prakash R, Raynal M, Singhal M. An adaptive causal ordering algorithm suited to mobile computing environments [J]. Journal of Parallel and Distributed Computing, 1997, 41(2): 190-204

[11] Hernández S P, Fanchon J, Drira K. The immediate dependency relation: an optimal way to ensure causal group communication [J]. Annual Review of Scalable Computing, 2004, 6(3): 61-79

[12] Zeng Fanyi, Li Sikun, Peng Yuxing, et al. A causal consistency mechanism in distributed virtual environment [J]. Journal of System Simulation, 2007, 19(3): 510-514 (in Chinese)
(曾凡益, 李思昆, 彭宇行, 等. 一种分布式虚拟环境中的因果一致性控制方法[J]. 系统仿真学报, 2007, 19(3): 510-514)

[13] Zhou S P, Cai W T, Turner S J, et al. Critical causal order of events in distributed virtual environments [J]. ACM Trans on Multimedia Computing, Communications and Applications, 2007, 3(3): Article 15

[14] Lamport L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565

[15] Kshemkalyani A D, Singhal M. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation [J]. Distributed Computing, 1998, 11(2): 91-111

[16] Dabek F, Cox R, Kaashoek F, et al. Vivaldi: A decentralized network coordinate system [C] //Proc of Special Interest Group on Data Communication. New York: ACM, 2004: 426-437

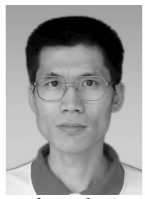
[17] Agarwal S, Lorch J R. Matchmaking for online games and other latency-sensitive P2P systems [C] //Proc of Special Interest Group on Data Communication. New York: ACM, 2009: 315-326

[18] Zhang Y, Zhou Z, Wu W. A hierarchical time management mechanism for HLA-based distributed virtual environment [J]. Journal of Computational Information Systems, 2006, 1(2): 7-15

[19] Zhao Qinping, Zhou Zheng, Lü Fang. Algorithm of simulation time synchronization over large-scale nodes [J]. Science in China Series F: Information Sciences, 2008, 51(9): 1239-1255

Zhou Hangjun, born in 1979. PhD from the College of Computer, National University of Defense Technology, Changsha, China. His current research interests include distributed virtual environments, cloud computing, Internet QoS routing, distributed network programming.

Zhang Wei, born in 1982. Assistant professor in the College of Computer, National University of Defense Technology, Changsha, China. His current research interests include distributed virtual environments, distributed circuit simulation, parallel and distributed systems(zw_nudt@163.com).



Peng Yuxing, born in 1963. Professor and PhD supervisor in the College of Computer, National University of Defense Technology, Changsha, China. His current research interests include distributed virtual environments, cloud computing and P2P networking(dvetsc@gmail.com).

Li Sikun, born in 1941. Professor and PhD supervisor in the College of Computer, National University of Defense Technology, Changsha, China. His current research interests include virtual reality and virtualization, embedded system design methodology(skli@nudt.edu.cn).