中国科学 E辑：信息科学 2008 年 第 38 卷 第 8 期：1139 ~ 1156
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE IN CHINA PRESS

大规模分布节点的仿真时间同步算法

赵沁平[①②], 周忠[①②*], 吕芳[①②]

① 虚拟现实技术与系统国家重点实验室(北京航空航天大学), 北京 100083;
② 北京航空航天大学计算机学院, 北京 100083
* E-mail: zz@vrlab.buaa.edu.cn

收稿日期: 2008-01-04; 接受日期: 2008-05-07
国家自然科学基金(批准号: 60603084)和中国高技术研究发展计划(批准号: 2006AA01Z331)资助项目

摘要 分布式仿真中难于实现统一的全局物理时钟, 且节点之间的传输存在抖动, 因此为了保证各仿真节点的事件一致性, 需要进行仿真时间同步. 论文对大规模分布节点的时间同步算法开展研究, 对 HLA 标准中定义的 LBTS 算法进行性质分析, 给出分组管理的 LBTS 计算模型. 针对现有时间同步算法将可靠控制报文作为默认前提的问题, 给出了时间控制报文可靠性定理, 证明了只有制约系统推进的一部分报文需要可靠, 该默认前提并非必要. 在此基础上, 设计了 MCTS(multi-node coordination time synchronization)算法, 根据时间控制报文可靠性定理引入 IP 组播来处理控制报文的传输, 提高了节点的处理效率, 大大降低了控制报文的带宽开销. 实验结果表明, 该算法在节点时间推进速度及网络带宽占用量方面优于同类算法, 1000 个节点的时间同步推进速度可达到 50 次/s, 相当于同类系统中 100 个节点时的推进速度.

关键词
分布式仿真
时间同步
多点协调
LBTS
组播

事件的一致性是分布式仿真的基本要求, 而在实际仿真过程中会出现不一致的现象, 如网络游戏或仿真中常见的“开火—爆炸”颠倒问题. 该问题的出现是由于在分布式系统中, 各个节点分布于不同的物理位置, 它们之间通过网络进行通讯. 由于网络传输时延不可确定加之各个分布节点处理能力的不同, 节点对事件的处理时间无法预计. 同时, 由于各个节点地理分散, 较难实现一个统一的全局物理时钟对它们进行同步. 通过用仿真时间对真实世界中的物理时间进行仿真, 以时间的一致性达到事件的一致性是一种有效的解决方法. 因此, 时间同步一直是分布式仿真领域的重点研究问题. 时间同步, 也称“时间管理”[1], 管理分布式仿真中各节点消息的产生和接收, 并协调不同的时间推进方式, 保证仿真过程的逻辑正确性. 时间管理是分布式仿真领域研究的重点和难点[2,3].

现有的时间管理算法存在时间推进效率低及网络带宽占用量大的不足[4]. 针对这个问题,

本文研究了分布式仿真中时间管理算法节点协调推进的性质, 给出了分组管理的LBTS计算模型, 分析了节点对不同控制报文的可靠性要求, 提出了控制报文可靠性定理, 进而提出了多点协调的仿真时间同步算法. 并将该算法应用于BH RTI时间管理服务, 对其进行实验对比分析, 结果表明, 该算法在时间推进速度和网络带宽占用量方面优于同类系统, 有利于仿真规模的扩展.

本文的组织如下: 第 1 节介绍了国内外相关研究现状; 第 2 节分析了时间管理算法计算节点安全推进时间上限的性质, 并给出了一种确定该上限值的计算模型; 第 3 节论证了时间同步协调过程中不同控制报文的可靠性要求, 提出了多点协调的仿真时间同步算法; 第 4 节给出了该算法应用于 BH RTI 时间管理服务的对比实验验证与结果分析; 第 5 节总结全文.

1 相关研究

分布式系统中的时间同步的目的是为了保证事件的一致性, Lamport算法以时间戳传递性递增的方式保证了因果性事件在各节点的处理一致性, 向量时钟在它的基础上扩展了多节点同时产生因果性事件的处理, 但它们都不能处理并发事件产生与处理的一致性. 分布式仿真中, 各节点需要并发产生大量事件, 严格来说, 事件在不同节点上的产生顺序要求绝对的一致性, 而不是仿真时间是否和真实时间成比例, 因此也不能仅依赖统一的物理时钟为参照. 因此现有的分布式仿真中一般通过仿真时间的方法来进行这种更为严格的时间同步. 时间管理最早源于并行离散事件仿真PDES(parallel discrete event simulation), 以并行的方式来处理仿真时间的一致性. 分布式仿真中的时间管理是在PDES时间管理基础上发展而来的, 通过网络传输来协调时间的同步推进[1,2].

早期经典的Chandy/Misra算法[5]中, 消息按时间戳非递减的顺序发送给逻辑进程LP, LP将消息存储在队列中, 每次先处理时戳最小的消息, 该算法可以保证进程本地的事件一致性, 但是易产生死锁. 为了解决Chandy/Misra算法的死锁问题, Chandy, Misra等又提出了Null Message(空消息)法. 逻辑进程LP_A对另一个逻辑进程LP_B发送Null Message进行声明, 表示LP_A不会再发送时戳小于某一未来时间的消息. 空消息法的思想引出了前瞻值(Lookahead)概念[2], 被后来大部分的时间同步算法采用.

现有的时间管理算法主要是通过确定时间推进区间来保证事件一致性, 这一类算法通过LP之间发送的时间信息来确定时间推进的范围, 保证时间推进中各节点不会收到过时的消息, 从而避免系统中出现不一致现象. 其重点在于时间推进区间的计算, 大多数算法中称该区间的边界值为时戳下限(lower bound time stamp, LBTS). HLA标准中还对时间管理中的LBTS和Lookahead的概念进行了定义. LBTS是节点的最大安全时间推进值, 将来不会再接收到时间戳小于该值的消息. Mattern[6]提出了通过分布式“快照”计算LBTS的方法, 该方法为每个LP维护了一个计数器, 其值为LP发送消息的数量减去其收到消息的数量, 当计数器值为 0 的时候, 发起LBTS的计算. Fredrick提出的LBTS计算方法, 通过各个仿真程序处在不同状态时的输出时间来计算该进程的LBTS值. 时间推进区间法可以比较有效地进行时间一致性管理, 在分布式仿真领域广泛使用, 并被HLA标准所确定, 但LBTS的计算要求集中实时维护各节点的仿真时间, 复杂度高, 不利于系统规模的扩展.

其他的时间同步算法还有Fujimoto[7]提出的近似时间(approximate time)的方法, 在分布式仿真的时间管理中采用时间段代替时间点来增加仿真的并行度, 进而对于小前瞻值或零前瞻值的分布仿真提高时间推进效率, 但是该方法存在时间段取值难于确定的问题, 其大小必须根据具体的仿真模型而定. Lee[8]提出了基于因果序(causal order)的时间管理模型, 以提高分布仿真中仿真时间推进效率, 但该方法不能处理非因果关系的事件序列.

在现有的各类时间同步算法[6~10]中, 将各节点之间进行可靠的控制信息交换作为缺省的必要条件, 这样虽然可以保证事件的一致性, 但是存在时间推进效率低及网络带宽占用量大的不足. 本文对分布式系统中仿真时间同步进行研究, 在时间推进区间法及前瞻值思想的基础上提出了多点协调的仿真时间同步算法MCTS, 并证明了只有部分信息需要可靠. 将该算法应用于BH RTI的时间管理服务, 进行了实验验证. 实验结果表明, 该算法能够提高时间推进速度并减少了网络带宽占用量, 优于国际上同类RTI系统的时间管理算法, 有利于仿真规模的扩展.

2 分组管理的 LBTS 计算模型

2.1 LBTS 概念

分布式仿真中, LBTS 是指节点能够安全推进的最大仿真时间. 为了保证仿真事件的一致性, 要求一个节点在仿真的任何时刻不能接收到一个过时(即比节点当前仿真时间小)的消息. 因而, 在节点推进时, 必须保证节点的时间推进不能超过 LBTS 这个界值, 否则在未来的仿真时间里有可能接收到一个过时的消息. 正确计算 LBTS 是关系到整个时间同步算法的核心问题, 分布式仿真中广泛采用了如下计算公式来计算节点 i 的 LBTS:

$$\mathrm{LBTS}_i = \min\{T(j)+L(j)\},\ i \neq j, \tag{1}$$

其中j表示i之外的任意节点, $T(j)$表示节点j的当前仿真时间, $L(j)$表示节点j的前瞻值(Lookahead). 前瞻值代表了仿真节点对其自身产生事件的预测能力, 即节点可以预测未来一段时间以后的事件, 保证在Lookahead时间内不会产生新的事件. 这样各个节点可以使用Lookahead将自己产生事件的时戳情形更早地通知给其他节点, 以加快系统的并发处理. Fujimoto等在DMSO于 1996 年发布的HLA时间管理草案[11]中首先给出了LBTS和公式(1)的定义, Fujimoto等在RTI F.0 的设计中也用到了公式(1)[12].

2.2 LBTS 性质分析

基于 LBTS 的时间管理算法要求每个仿真节点按照时间递增的顺序协调推进, 不能处理“过去”的消息, 因此, 节点的每一次推进都要确保该次请求之后, 不会再有时间戳小于该次请求时间的消息到来. 其中每个节点都具有预测能力, 可以向其他节点做出承诺在时间长度等于 Lookahead 的时间间隔内, 不会产生新的事件. 根据其他节点的承诺时间, 每个节点都可以计算出一个时间值, 并且可以推进到小于该时间值的任一时间点, 不会收到过时的消息, 这个时间值就是 LBTS. 同时, 节点产生的事件的时戳也有具体的要求, 即事件的时戳必须大于或者等于该节点的当前逻辑时间与 Lookahead 值之和.

设有n个仿真节点N_1, N_2, … , N_n, 节点N_i在当前时刻的仿真时间为t, 接收事件为E_i, 其时戳值为$(E_i.\eta)$. 为分析节点逻辑时间与事件时戳之间的关系, 我们进行如下定义:

1) 承诺逻辑时间(promised logical time, PLT). 节点当前逻辑时间与前瞻值之和, 表示为PLT =t+Lookahead, 它是在仿真时刻t, 节点向其他节点承诺的未来事件的时间戳的下限值, 保证以后所有事件的时间戳都不会小于该值, 因此任一事件E_i的时戳$E_i.\eta$必须满足 $E_i.\eta \geqslant$ PLT.

2) 安全事件集合(safe event set, SES). 节点接收到的所有事件的一个子集, 是当前逻辑时间t的函数, 表示为SES(t)= $\{E_i | E_i.\eta \leqslant t\}$, 该集合中的事件可以被安全地处理.

可以证明, LBTS 具有以下性质:

定理 1 节点的时戳下限取值为其他各节点承诺值的最小值, 可以保证不会收到过时的消息.

证明 设有n个仿真节点N_1, N_2, …, N_n, 令N =$\{N_i\}$, i =1, …, n, 它们的当前逻辑时间为t_1, t_2, … , t_n, 承诺值为σ_1, σ_2, … , σ_n, 下一事件的时戳值分别为η_1, η_2, … , η_n.

根据承诺逻辑时间的定义, 有$\eta_1 > \sigma_1$, $\eta_2 > \sigma_2$, $\eta_3 > \sigma_3$, … , $\eta_n > \sigma_n$, 考虑到命题不是针对特定节点, 将它们按承诺时间递增顺序排列, 设其承诺逻辑时间大小关系为$\sigma_1 < \sigma_2 < \sigma_3, \ldots, < \sigma_n$, 因此, $\sigma_1 < \eta_1$, $\sigma_1 < \eta_2, \ldots, \sigma_1 < \eta_n$. 下面对节点的承诺时间值为非最小和最小两种情形来进行讨论:

1) 对于N_i, 当 $2 \leqslant i \leqslant n$时, 承诺时间值为非最小. 其他PLT的最小值为$\sigma_1$.

设节点推进到时间t_i', $t_i' < \sigma_1$, 那么有$t_i' < \eta_1$, $t_i' < \eta_2, \ldots, t_i' < \eta_n$.

可知, 当节点的承诺时间值为非最小时, 时戳下限取值为其他节点承诺值的最小值, 不会收到过时的消息.

2) 对于N_i, 当i=1 时, 承诺时间值为最小. 其他PLT的最小值为σ_2.

设其推进到t_1', $\sigma_1 < t_1' < \sigma_2$, 相应地, 其承诺逻辑时间变为$\sigma_1'$, $\sigma_1 < \sigma_1'$, 当前, 各个节点可以发送的消息的时间戳分别为$\eta_1, \eta_2, \ldots, \eta_n$.

由承诺逻辑时间的定义, 可得$\eta_1 > \sigma_1', \eta_2 > \sigma_2, \eta_3 > \sigma_3, \ldots, \eta_n > \sigma_n$, N_1可以收到来自其他节点的消息, 时戳分别为$\eta_2, \eta_3, \ldots, \eta_n$, $t_1' < \sigma_2 < \eta_2$, $t_1' < \sigma_2 < \sigma_3 < \eta_3, \ldots, t_1' < \sigma_2 < \sigma_3 < \cdots < \sigma_n < \eta_n$.

可知, 当节点的承诺时间值为最小时, 时戳下限取值为其他节点承诺值的最小值, 不会收到过时的消息.

综合以上 1)和 2), 定理 1 成立. 证毕

2.3 分组管理的 LBTS 计算模型

在定理 1 的基础上, 我们可以给出分组管理的 LBTS 计算模型. 将参与计算的节点进行分组, 如果节点的时间信息发生改变导致 LBTS 值发生改变, 那么首先在组内进行计算, 得出该组的最小时间值和次小时间值, 然后对各组的最小时间值和次小时间值组成的集合进行全局计算, 得出全局的最小时间值与次小时间值. 最终根据节点的时间推进状态以及全局时间求得各个节点的 LBTS 值. 计算方法模型可以表述如下:

设系统中存在m个协调节点$\boldsymbol{C}$=$\{C_1, C_2, C_3, \ldots, C_m\}$, 每个协调节点负责$n_h(1 \leqslant h \leqslant m)$个仿

真节点的时间同步协调. 则所有仿真节点分作m组, 每组数量为n_h个, 对任一节点N拥有时间值t和承诺时间值σ, 可得集合$\Omega_1, \Omega_2, \ldots, \Omega_m$:

$$\Omega_1 = \{\sigma_{1,1}, \sigma_{1,2}, \sigma_{1,3}, \ldots, \sigma_{1,n1}\},$$
$$\Omega_2 = \{\sigma_{2,1}, \sigma_{2,2}, \sigma_{2,3}, \ldots, \sigma_{2,n2}\},$$
$$\vdots$$
$$\Omega_m = \{\sigma_{m,1}, \sigma_{m,2}, \sigma_{m,3}, \ldots, \sigma_{m,nm}\}.$$

对于节点集合Ω_k, $1 \leqslant k \leqslant m$, 通过各个节点的承诺时间值$\sigma$可以确定协调节点$C_k$的两个变量值$(\lambda_k, \delta_k)$, 它们分别为组内节点承诺时间值$\sigma$的最小值和次小值, 计算公式表示为

$$\lambda_k = \sigma_{ki}, \text{ 满足}((\sigma_{ki} \in \Omega_k) \wedge (\forall j(\sigma_{ki} \leqslant \sigma_{kj}));$$
$$\delta_k = \sigma_{ki}, \text{ 满足}((\sigma_{ki} \in \{\Omega_k - \{\lambda_k\}\}) \wedge (\forall j(\sigma_{ki} \leqslant \sigma_{kj})).$$

进而得到集合Φ, 表示如下:

$$\Phi = \{\lambda_1, \delta_1, \lambda_2, \delta_2, \lambda_3, \delta_3, \ldots, \lambda_m, \delta_m\}.$$

令$\mu_k = \lambda_k$, $\mu_{k+m} = \delta_k$, 可得$\Phi = \{\mu_1, \mu_2, \mu_3, \mu_4, \ldots, \mu_{2m-1}, \mu_{2m}\}$.

对于集合Φ, 可以确定两个变量值(ψ, ξ), ψ和ζ分别为集合Φ中的最小值和次小值, 它们的计算公式为

$$\psi = \mu_k, \text{ 满足}((\mu_k \in \Phi) \wedge (\forall j(\mu_k \leqslant \mu_j));$$
$$\xi = \mu_k, \text{ 满足}((\mu_k \in \{\Phi - \{\psi\}\}) \wedge (\forall j(\mu_k \leqslant \mu_j)).$$

通过全局计算的结果, 得到承诺时间最小的节点集合$\Phi_{\min}$以及变量值(λ_k, δ_k)最小的协调节点集合$\boldsymbol{C}_{\min}$,

$$\Phi_{\min} = \{N_i \mid \sigma_i = \psi\}, \quad \boldsymbol{C}_{\min} = \{C_k \mid \lambda_k = \psi\}.$$

最终确定的变量值ψ和ξ就是要求的各个节点的可以推进的时间的上限, 对于承诺时间值最小的节点, 其上限为ξ, 其他节点的上限为ψ. 因此可以得到各个节点的 LBTS 取值为

$$\text{LBTS}_i = \begin{cases} \psi, & N_i \notin \Phi_{\min}, \\ \xi, & N_i \in \Phi_{\min}. \end{cases}$$

通过这种分组管理的 LBTS 计算模型, 可以方便地将时间信息的维护和计算分配到不同的节点上, 减轻节点的负载, 同时由于时间信息一致性维护的区域缩减到组的范围内, 以组为单位进行管理, 节点之间的通信量也会降低.

3 MCTS 算法模型

分布式仿真中, 各节点通过相互发送消息报文来进行信息的传送, 从而进行仿真时间推进的协调. 以往的一些时间管理算法都存在“控制报文必须可靠”的默认前提, 不允许控制报文发生丢失. 随着仿真规模的扩大, 这个限制对网络资源提出了较高的要求, 而且随着仿真节点的增多, 会造成节点的时间推进速度迅速下降. 通过对节点时间同步过程进行分析, 发现对于每个节点及其所发送的控制报文, 具有组通讯的特点, 因此可以考虑将组播引入到时间同步算法中, 从而降低网络开销, 提高时间推进的效率. 但是由于“控制报文必须可靠”的前提, 使得不可靠的 IP 组播不能被使用. 在节点同步过程中, 是否所有的节点都要可靠地接收所有的报文, 是本节所要研究的重点问题.

3.1 控制报文可靠性定理

在基于分组管理的 LBTS 计算模型的时间同步系统中, 节点之间通过时间同步控制报文来进行同步协调, 为了进行节点对报文的可靠性要求分析, 首先将这些报文进行分类. 如图 1 所示, 设系统中有 n 个仿真节点以及一个协调节点 C, 它们之间相互发送报文来传递信息. 这些报文可以分作两类: 一类是由仿真节点发送至协调节点的包含节点承诺时间σ以及分组计算得到的λ, δ信息的报文, 由于λ和δ的值是由σ值计算而得, 因此我们简称该类报文为σ报文. 另一类是由协调节点发送至仿真节点的包含了用于计算 LBTS 的ψ值和ξ值的信息报文, 我们简称该类报文为ψ-ξ报文. 两类报文在节点之间的收发关系如图 1 所示.

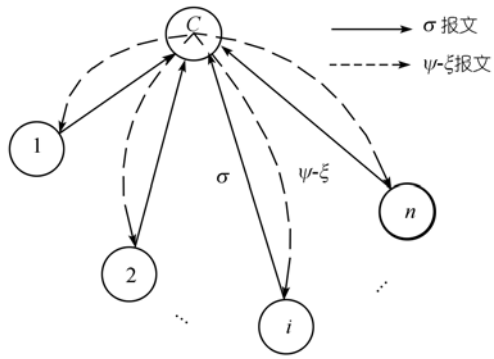

图 1　节点之间报文分类

仿真过程中, 每个节点仿真时间推进的大小及前瞻值是不同的, 根据它们承诺时间为: 最小、次小、既不是最小也不是次小, 可以把它们分为最小节点 MIN, 次小节点 SEC 以及其他的普通节点 GEN 共 3 类, 表示为 NodeType = enum{MIN, SEC, GEN}. 下面我们将研究报文传输的可靠性与仿真能否正确推进之间的关系.

定理 2　σ报文可靠传输是仿真正确推进的必要条件.

证明　设n个仿真节点N_1, N_2, … , N_n的当前仿真时间分别为t_1, t_2, … , t_n, 承诺时间PLT_1, PLT_2,…, PLT_n分别为σ_1, σ_2, … , σ_n. 为了方便叙述, 假设$t_1 < t_2 < t_3 < \cdots < t_n$, $\sigma_1 < \sigma_2 < \sigma_3 < \cdots < \sigma_n$.

以 4 个节点为例进行证明, 如图 2(a)所示, 其中 3 个节点N负责具体的执行逻辑, 1 个节点C对它们的时间推进进行协调. 其中$\mathrm{NodeType}(N_1)$= MIN, $\mathrm{NodeType}(N_2)$= SEC, $\mathrm{NodeType}(N_3)$= GEN, 它们的推进情形、当前逻辑时间以及承诺时间如图所示. 3 个节点将自身的σ值通过报文发送给C, 在没有产生丢包的情形下, C计算得出的正确同步结果是{σ_1, σ_2}, 下面按照发送报文节点的NodeType分别为MIN, SEC, GEN共 3 种情形进行讨论.

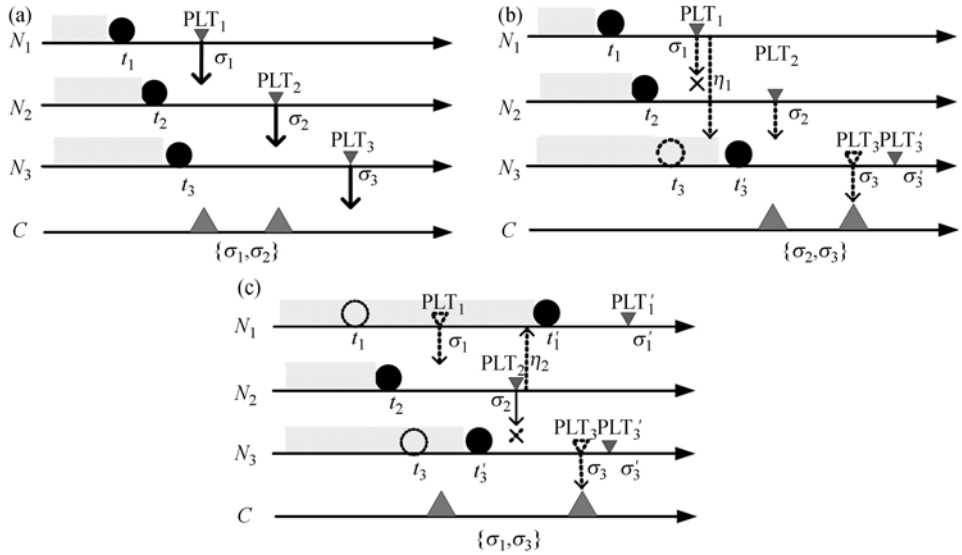

图 2 σ 报文丢失的 3 种情形

(a) 节点发送σ报文前的推进状态; (b) NodeType 为 MIN 的节点的σ报文丢失; (c) NodeType 为 SEC 的节点的σ报文丢失

情形 1 NodeType 为 MIN 的节点的σ报文丢失.

如图 2(b)所示, 假设N_1向协调节点C发送的σ报文在传输过程中丢失, 因此, 在C处计算并返回给各节点的结果为$\{\sigma_2, \sigma_3\}$. 此时, N_3的安全推进时间上限$LBTS_3=\sigma_2$, 可以推进到小于σ_2的任一时间值. 假设N_3推进到t_3', $\sigma_1 < t_3' < \sigma_2$, N_1这时没有进行推进, 因此其PLT值σ_1保持不变. N_1发送了一个时间戳为η_1的消息, 其中$\sigma_1 < \eta_1 < t_3'$, N_3收到来自N_1的消息, $\eta_1 < t_3'$, 收到一个过时的消息, 出现不一致. 因此, NodeType为MIN的节点发出的σ报文必须保证可靠到达.

情形 2 NodeType 为 SEC 的节点的σ报文丢失.

如图 2(c)所示, 假设N_2向协调节点C发送的σ报文在传输过程中丢失, 因此, 在C处计算并返回给各节点的结果为$\{\sigma_1, \sigma_3\}$. 此时, 各个节点的LBTS分别为: $LBTS_1=\sigma_3$, $LBTS_2=\sigma_1$, $LBTS_3=\sigma_1$, 它们可以推进到小于其LBTS的任一未来时间值. 假设N_1推进到t_1', $\sigma_2 < t_1' < \sigma_3$, 相应的, 其PLT值变为$\sigma_1'$, N_2这时没有进行推进, 因此PLT_2保持不变, N_2发送了一个消息, 时间戳为η_2, 其中$\sigma_2 < \eta_2 < t_1'$, N_1收到来自N_2的消息, $\eta_2 < t_1'$, 收到一个过时的消息, 出现不一致. 因此, NodeType为SEC的节点发出的σ报文必须保证可靠到达.

情形 3 NodeType 为 GEN 的节点的σ报文丢失.

当普通节点发送的报文丢失时, 理论上分析, 不会产生事件的不一致现象. 但是在实际的推进过程中, 每个节点都有可能成为最小节点或者次小节点. 并且在节点本地, 时间信息没有保证一致之前, 每个节点不能判断出当前自己是最小节点还是次小节点. 因此, 节点的NodeType 为 GEN 的节点发出的σ报文也要保证可靠到达.

综合情形 1~3, 定理 2 成立. 证毕.

通过以上的证明可知, 仿真推进过程中, 用于传送各个节点时间信息的报文处于比较重要的地位, 如果不能获取完全、准确的信息, 就无法正确进行全局时间同步. 因此我们可以得出, 在分组管理的 LBTS 计算模型中, 为了保证仿真的正确推进, 必须要保证σ报文传输的可靠.

定理 3 最小节点可靠接收ψ-ξ报文是仿真正确、及时推进的必要条件, 其他节点可靠接收ψ-ξ报文不影响时间推进的正确性, 是不必要条件.

证明 设各个节点的状态及属性值如图 3(a)所示, 当前 3 个节点N_1, N_2, N_3, 它们请求推进的时间分别为t_1, t_2, t_3, LBTS线表示 3 个节点的当前安全时间推进上限, LBTS = $\sigma < t_1 < t_2 < t_3$, 可知 3 个节点均处于时间推进等待状态. 其中NodeType(N_1)=MIN, NodeType(N_2)=SEC, NodeType(N_3)=GEN. 协调节点C接收到各个节点发送的一致性消息, 计算自身的$\{\lambda, \delta\} = \{\sigma_1, \sigma_2\}$, 下面根据接收节点的NodeType分别为MIN, SEC, GEN共 3 种情形来说明其对ψ-ξ报文的可靠性要求.

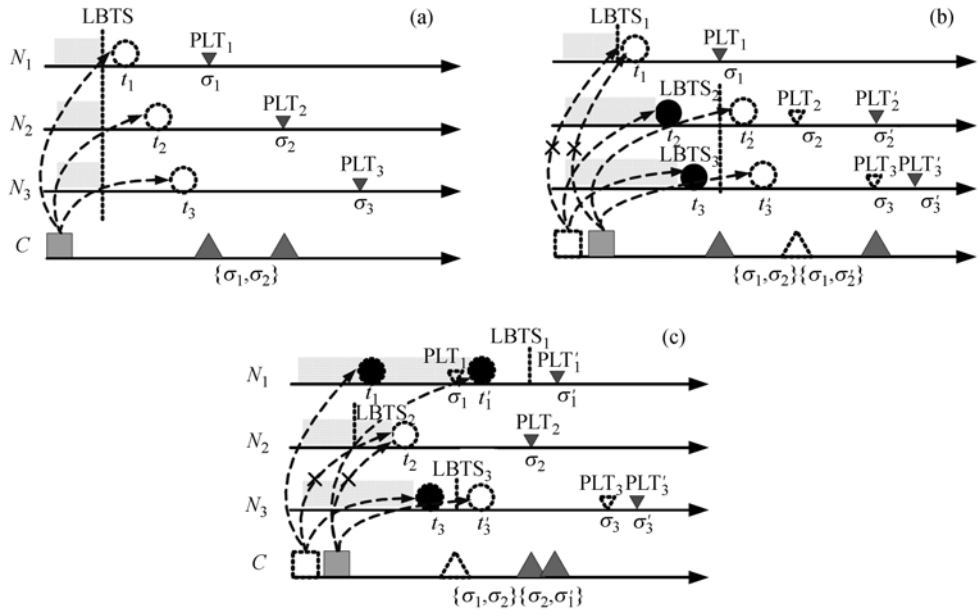

图 3 ψ-ξ报文丢失的 3 种情形

(a) 节点的当前推进状态; (b) NodeType 为 MIN 的节点的ψ-ξ报文丢失; (c) NodeType 为 SEC 的节点的ψ-ξ报文丢失

情形 1 NodeType 为 MIN 的接收节点的ψ-ξ报文丢失.

如图 3(b)所示, 协调节点C发送给各个节点的ψ-ξ报文, 发给N_1的报文发生丢失, N_2和N_3收到报文后, 计算得出$\mathrm{LBTS}_2 = \mathrm{LBTS}_3 = \sigma_1$. 由于$N_1$没有接收到该报文, 因此其$\mathrm{LBTS}_1$保持不变, 仍为$\sigma$, 因为$\mathrm{LBTS}_2 > t_2$, $\mathrm{LBTS}_3 > t_3$, 所以N_2和N_3的时间推进被允许. 因为$\mathrm{LBTS}_1 = \sigma < t'_1$, 因此, N_1继续保持时间等待状态, 其PLT_1值也保持不变. N_2和N_3进行下一次推进并发送PLT值, 推进时间分别为t'_2, t'_3, 承诺逻辑时间 PLT'_2, PLT'_3 分别为 σ'_2, σ'_3, 协调节点C计算进行计算, 得$\{\lambda, \delta\} = \{\sigma_1, \sigma'_2\}$, 再次向各节点发送协调控制报文, 如果发给$N_1$节点的$\psi$-$\xi$报文再次丢失, 那么$N_1$

将仍然保持推进等待状态, 这时N_2, N_3计算得出$LBTS_2 = LBTS_3 = \sigma_1 < \sigma_2' < \sigma_3'$, 不能向前推进, 因此, 3 个节点将都处于等待状态, 系统发生死锁, 无法推进.

可见, 要使各个节点可以正常推进, NodeType为MIN的节点必须接收到ψ-ξ报文, 否则其他节点最多只能推进到σ_1.

情形 2 NodeType 为 SEC 的接收节点的ψ-ξ报文丢失.

如图 3(c)所示, 协调节点C发送给各个成员节点的ψ-ξ报文, 发给N_2的报文发生丢失, N_1, N_3收到报文后, 计算得出$LBTS_1=\sigma_2$, $LBTS_3=\sigma_1$. 由于N_2没有接收到该报文, 因此其$LBTS_1$保持不变, 因为$LBTS_1 > t_1$, $LBTS_3 > t_3$, 因此N_1, N_3时间推进允许, 而$LBTS_2 < t_2$, 因此, N_2继续保持时间等待状态, 其PLT_2值保持不变仍为σ_2. N_1, N_3进行下一次推进, 推进时间分别为t_1', t_3', 承诺逻辑时间值分别为σ_1', σ_3', $\sigma_2 < \sigma_1' < \sigma_3'$, N_1, N_3将新的PLT值发送给C, C收到时间信息报文后进行计算, 得$\{\lambda, \delta\}=\{\sigma_2, \sigma_1'\}$, 并向各个节点再次发送报文, 这时, N_2成为各个节点中推进最慢的节点, NodeType变为MIN.

可见, NodeType 为 SEC 的节点, 如果其接收的ψ-ξ报文丢失, 那么几次推进后, 它的 NodeType 将成为 MIN.

情形 3 NodeType 为 GEN 的接收节点的ψ-ξ报文丢失.

从对情形 2 的分析也可以得出, 对于NodeType为GEN的节点N_3, 如果连续收不到ψ-ξ报文, 那么几次推进以后, 它将NodeType变为SEC的节点, 进而成为NodeType为MIN的节点.

综合情形 1~3, MIN 节点接收ψ-ξ报文丢失会导致死锁, SEC, GEN 节点接收ψ-ξ报文丢失会影响推进速度, 但不影响推进正确性, 定理 3 成立. 证毕.

通过以上证明可知, 时间同步过程中, 对于协调节点返回给被协调节点的控制信息, 我们不需要保证所有的这类信息都被完全可靠地接收. 只要优先保证那些制约系统时间推进的节点(即当前 PLT 最小的那些节点)能够及时收到这类消息即可. 因此可以得出, 在分组管理的 LBTS 计算模型中, 最小节点可靠接收ψ-ξ报文是分组管理的 LBTS 计算模型中仿真正确、及时推进的必要条件, 其他节点可靠接收ψ-ξ报文则是不必要条件.

综合定理 2 和 3 可得, 时间同步过程中, 仿真节点所接收的控制报文不一定必须要求全部可靠, 仿真的推进只取决于关键节点的报文能否正确到达, 因此, 只要优先保证那些制约系统推进的节点及时收到重要的控制报文, 那么整个仿真就可以进行时间推进. 将以上的分析总结为时间同步控制报文可靠性定理, 具体表述如下:

时间同步控制报文可靠性定理 基于分组管理的 LBTS 计算模型的仿真时间同步系统中, 只要保证σ报文传输的可靠性以及最小节点可靠接收ψ-ξ报文, 仿真可以正确进行时间推进.

证明 在基于分组管理的LBTS计算模型的仿真时间同步系统中, 保证了σ报文被可靠接收, 那么由成员节点发往协调节点的用于通知自身时间值的信息将被可靠接受, 这样协调节点可以获得完全的信息值进行 LBTS 值的计算, 因此用于全局控制的时间信息是准确的.

在时间同步过程中, 节点的推进受最小仿真节点(即时间推进最慢的节点)的制约, 最小节点不能推进, 整个仿真将不能及时推进. 保证最小节点可靠接收ψ-ξ报文, 那么最小节点将可以持续推进, 仿真推进也得到了保证.

而对于那些没有可靠接收到ψ-ξ报文的其他节点, 即使其出现不能推进的情形, 随着其他节点的推进, 它将成为最小的节点, 从而可靠地得到ψ-ξ报文, 可以继续推进.

综上所述, 只要保证σ报文传输的可靠性以及最小节点可靠接收ψ-ξ报文, 仿真可以正确进行时间推进. 证毕.

3.2 MCTS 算法模型设计

通过对分布式仿真节点 LBTS 性质以及各个节点对不同类型信息可靠要求的分析, 基于分组管理的 LBTS 计算模型及控制报文可靠性定理, 提出多点协调的仿真时间同步算法模型. 将各个分布的节点分作不同的角色, 每个节点依据其角色的不同, 只需负责系统中部分时间信息的维护与计算. 节点之间根据报文类型选择组播或可靠单播进行协调信息的传输, 通过协调组的同步协调, 达到节点的仿真时间同步.

3.2.1 节点角色划分

基于分组管理的 LBTS 计算模型, 首先将各个分布节点根据其职能的不同进行角色的划分, 分为 3 类: 成员节点、局部协调者节点和全局协调者节点. 每个节点根据其角色的不同, 只需维护整个时间同步中的部分时间信息, 节点与角色的对应关系如图 4 所示.

下面对 3 种角色节点进行说明:

1) 成员节点(member node, MN), 负责具体的执行逻辑, 拥有属性值分别为前瞻值 L、承诺逻辑时间 PLT、安全时间推进上限 LBTS 和安全事件集合 SES, 相关概念已在第 2 节给出.

2) 局部协调者节点(local coordinator node, LCN), 负责对其管理范围内的成员节点的时间同步进行协调, 拥有属性分别为局部最小时戳(local min timestamp, LMTS)、局部次小时戳(local second min timestamp, LSTS)和最小节点集合(min node set, MNS). LMTS及LSTS的值由该LCN协调范围内的成员节点计算而来, 可以对应于LBTS计算模型中的λ, δ, 对于任一局部协调者节点LCN_k, MNS对应于计算模型中的$\Phi_{\min}$. 在局部协调者范围内的成员节点的承诺时间值构成集合Ω_k, $\Omega_k = \{\mathrm{PLT}_{k1}, \mathrm{PLT}_{k2}, \mathrm{PLT}_{k3}, \dots, \mathrm{PLT}_{kn}\}$. 局部协调者节点内各个属性可以表示如下:

$$\mathrm{LMTS}_k = \mathrm{PLT}_{ki},\ \text{满足}((\mathrm{PLT}_{ki} \in \Omega_k) \wedge (\forall j\ (\mathrm{PLT}_{ki} \leqslant \mathrm{PLT}_{kj})),$$

$$\mathrm{LSTS}_k = \mathrm{PLT}_{ki},\ \text{满足}((\mathrm{PLT}_{ki} \in \{\Omega_k - \{\mathrm{LMTS}_k\}\}) \wedge (\forall j(\mathrm{PLT}_{ki} \leqslant \mathrm{PLT}_{kj})).$$

3) 全局协调者节点(global coordinator node, GCN), 负责对整个系统中的局部协调者的同步进行协调, 拥有属性分别为全局最小时戳(global min timestamp, GMTS)、全局次小时戳(global second min timestamp, GSTS)和最小局部协调节点集合(min local coordinator set, MLCS). GMTS及GSTS的值由该GCN协调的各个LCN计算而来, 可以对应于LBTS计算模型中的ψ和ξ, 对于全局协调者节点GCN, MLCS对应于计算模型中的$\boldsymbol{C}_{\min}$. 对于全局协调者节点GCN, 其协调范围内的所有局部协调者的局部最小时戳和局部次小时戳构成了集合Φ, $\Phi = \{\mathrm{LMTS}_1, \mathrm{LSTS}_1, \mathrm{LMTS}_2, \mathrm{LSTS}_2, \dots, \mathrm{LMTS}_m, \mathrm{LSTS}_m\}$. 令$\mathrm{LTS}_i = \mathrm{LMTS}_i$, $\mathrm{LTS}_{m+i} = \mathrm{LSTS}_i$, $1 \leqslant i \leqslant m$, 则集合$\Phi$还可以表示为$\Phi = \{\mathrm{LTS}_1, \mathrm{LTS}_2, \mathrm{LTS}_3, \mathrm{LTS}_4, \dots, \mathrm{LTS}_{2m-1}, \mathrm{LTS}_{2m}\}$. 全局协调者节点内各个属性表示如下:

$GMTS = LTS_k$, 满足$((LTS_k \in \Phi) \wedge (\forall j\ (LTS_k \leqslant LTS_j))$,

$GSTS = LTS_k$, 满足$((LTS_k \in \{\Phi - \{GMTS\}\}) \wedge (\forall j\ (LTS_k \leqslant LTS_j))$,

$MLCS = \{LCN_k | LMTS_k = GMTS\}$.

通过全局协调者节点的属性, 可以得到局部协调者节点的 MNS 集合, 表示如下:

$$MNS_k = \{MN_{ki} | PLT_{ki} = GMTS\},$$

从而得到 LBTS 的取值如下:

$$LBTS_{ki} = \begin{cases} GMTS, MN_{ki} \notin MNS_k, \\ GSTS, MN_{ki} \in MNS_k. \end{cases}$$

明确了各个角色节点的职能以及属性值, 图 4 示意了各种角色节点之间的关系. 可以看出成员节点与局部协调者之间, 局部协调者与全局协调者之间, 均为一对多的关系. 如图 4 和 5 所示, 其中, 局部协调者和全局协调者是作为协调节点, 对其范围内的各个节点的时间同步进行协调. 成员节点向其局部协调者发送σ报文, 局部协调者向全局协调者发送σ报文, 全局协调者计算出全局的最小和次小 PLT 时间. 发布则需要经过两级的协调, 全局协调者向局部协调者组播ψ-ξ报文, 局部协调者向其协调范围内的成员节点的通知可以有两种选择, 组播ψ-ξ报文或直接计算出它们最新的 LBTS 并分别单播发送.

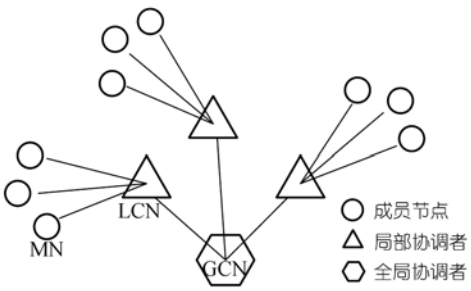

图 4 节点角色划分

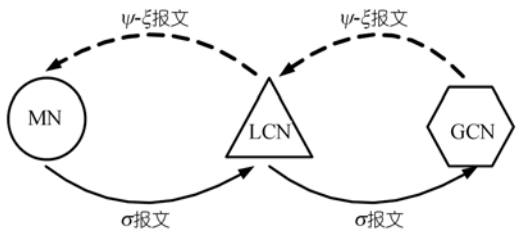

图 5 节点之间传输的时间报文类型

3.2.2 在时间报文传输中引入 IP 组播

仿真过程中, 各种角色节点之间通过相互发送时间报文来协调推进. 基于前面证明的控制报文可靠性定理, 可以根据报文-节点关系选择不同的节点之间的通讯策略. 对于可靠性要求高的报文, 采用可靠单播的通讯方式, 而对于可靠性要求不高的报文, 则可以采用 IP 组播, 利用组播来降低网络带宽资源开销, 提高组内的传输效率.

根据可靠性定理, σ报文必须保证可靠, MCTS 算法使用 TCP 进行σ报文的传输; ψ-ξ报文只要求最小成员接收可靠, MCTS 算法使用不可靠的 UDP 报文发送ψ-ξ报文, 并使用 TCP 向最小成员发送一份重复的ψ-ξ报文, 以处理可能的丢包. 算法的报文发送如图 6 所示, 其中实线表示采用 TCP 实现的可靠单播, 虚线表示不可靠的 UDP 组播.

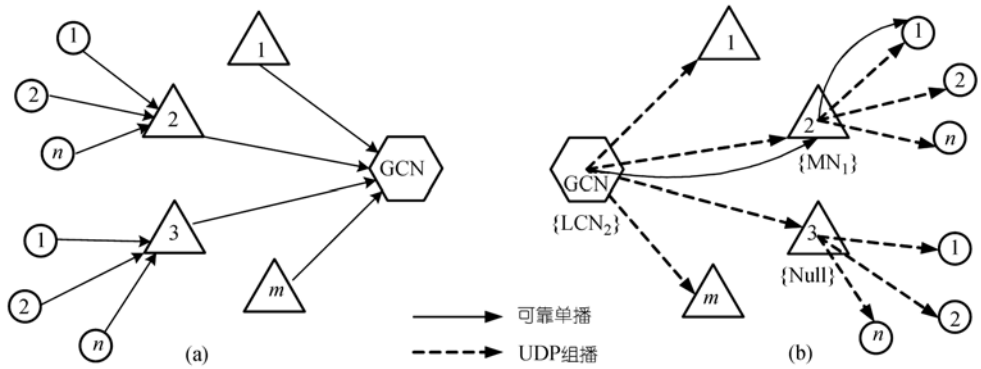

图 6　角色节点间的时间报文传输方式

(a) σ报文传输; (b) ψ-ξ报文传输

图6(a)是节点之间发送σ报文的示意图, 成员节点发往局部协调者以及局部协调者发往全局协调者的报文采用可靠单播的通讯方式. 图 6(b)是节点之间发送ψ-ξ报文的示意图, 将不可靠的 UDP 组播与可靠单播相结合, 全局协调者向局部协调者发送ψ-ξ报文时, 首先通过组播的方式进行发布, 然后通过本地维护的时间数据, 确定最小节点的集合, 向最小节点以TCP的方式发送协调控制报文. 局部协调者向其协调范围内的成员节点发布协调消息可以是类似的ψ-ξ报文组播流程, 也可以直接计算出节点的最新 LBTS 并分别单播发送.

根据定理 3, 需要确定可靠节点集合(reliable node set, RNS), 它是指协调节点所确定的需要发可靠消息的那些节点的集合. GCN需要在发布ψ-ξ报文时使用, LCN节点如果采用同样的流程也需要确定相应的 RNS. 可以表示为

$$\mathrm{RNS_{LCN}} = \{\mathrm{MN}_i \mid \mathrm{MN}_i \in \mathrm{MNSet} \wedge \mathrm{PLT}_i = \mathrm{GMTS}\},$$

$$\mathrm{RNS_{GCN}} = \{\mathrm{LCN}_k \mid \mathrm{LCN}_k \in \mathrm{LCNSet} \wedge \mathrm{LMTS}_k = \mathrm{GMTS}\},$$

MN_i为某一局部协调者上的第i个成员节点, MNSet为该局部协调者管理范围内所有成员节点集合, LCN_k为第k个局部协调者, LCNSet为所有局部协调者节点集合.

3.2.3　协调推进过程

基于以上定义, 3 种不同角色的节点之间的协调推进过程具体分为以下 6 个步骤.

步骤 1　成员节点MN_{ki}提出时间推进请求, 计算其PLT值, 向局部协调者节点LCN_k发送σ报文, 通知LCN_k自己当前的PLT.

步骤 2　LCN_k接收来自MN_{ki}的时间数据信息, 更新本地保存的MN_{ki}的PLT值, 判断更新的PLT_{ki}是否对LCN_k节点的局部特征值(LMTS_k, LSTS_k)产生了影响, 如果没有影响则不做处理, 继续等待下一报文, 否则根据自己所维护的其协调范围内的各成员节点(MN_{k1}, MN_{k2}, … , MN_{kn})的承诺逻辑时间值(PLT_{k1}, PLT_{k2}, … , PLT_{kn}), 计算(LMTS_k, LSTS_k), 发送σ报文给全局协调者节点GCN, 通知新的局部特征值.

步骤 3　GCN接收来自LCN_k的时间数据信息, 更新本地保存的LCN_k的特征值, 判断更新的(LMTS_k, LSTS_k)是否对GCN节点的特征值(GMTS, GSTS)产生了影响, 如果没有影响则不做

处理, 继续等待后续报文, 否则根据其上维护的系统中的所有局部协调者(LCN_1, LCN_2, … , LCN_m)的特征值{($LMTS_1$, $LSTS_1$), ($LMTS_2$, $LSTS_2$), … , ($LMTS_m$, $LSTS_m$)}计算得出全局特征值(GMTS, GSTS).

步骤 4 GCN计算得出新的全局特征值(GMTS, GSTS)之后, 向所有LCN节点组播ψ-ξ报文, 通知它们最新的全局时间信息, 然后确定集合RNS_{GCN}, 对任一节点$LCN \in RNS_{GCN}$, 发送TCP的ψ-ξ报文.

步骤 5 对于任一局部协调者节点LCN_k, 如果$LCN_k \in RNS_{GCN}$, 那么它可能会收到组播和TCP两个协调信息相同的报文, 因此节点首先要进行是否收到重复报文的判断, 如果收到了重复报文, 则丢弃. 接下来LCN_k对收到的ψ-ξ信息进行处理, 可以有两种方法: 1) 向成员节点组播ψ-ξ报文, 通知它们最新的全局时间信息, 然后确定集合RNS_{LCN}, 对任一节点$MN \in RNS_{LCN}$, 发送TCP的ψ-ξ报文; 2) 根据全局特征值(GMTS, GSTS)计算其管理范围内各MN的LBTS值, 向其协调范围内的MN节点分别点播发送它们最新的LBTS值. 第 1 种方法进一步降低了带宽需求, 但需要更多的动态组播地址; 第 2 种方法实现简单, 而且可以减轻MN的计算. 考虑到实现工作量, 我们在实现中采用了第 2 种方法.

步骤 6 节点MN_{ki}收到来自LCN_k的协调报文, 更新自身的LBTS值, 如果请求推进的时间超出了安全推进范围, 则继续等待; 否则, 进行时间推进并处理节点的安全事件集合, 然后继续下一次推进.

4 实验与结果分析

将提出的多点协调的仿真时间同步算法应用于 BH RTI(http://www.hlarti.com)的时间管理服务, 实现后的 BH RTI 时间管理服务包括两部分: 一部分是时间管理服务器 TMServer, 对应于本文算法中的全局协调者节点; 另一部分是分布的 RTI 部件 RtiExec, 对应于本文算法中的局部协调者节点, 在此基础上进行了实验验证.

4.1 实验环境

实验的网络环境、主机环境以及软件如表 1 所示.

表 1 实验环境表

网络环境	100 M 以太网, 华为 Quidway S3928
主机环境	Pentium 4 CPU 3.40 GHz, 内存 1 GBytes, Windows XP, 共 11 台
软件	RTI 软件: BH RTI 2.3, DMSO RTI 1.3NGv6, Mak RTI 3.0, pRTI 1516LE
	测试程序: DMSO Benchmark(美国国防部建模与仿真办公室发布的 RTI 性能测试标准程序)
	网络带宽测试软件: IRIS v4.07.1 (http://www.eeye.com/)

实验对基于本文提出的算法实现的 BH RTI 时间管理服务进行验证, 同时还测试了同等环境下国际同类 RTI 系统的时间管理服务, 包括 DMSO RTI1.3NG-V6, Pitch pRTI1516LE 和 Mak RTI 3.0. 实验所用主机中, 选取一台作为服务主机, 其他作为仿真主机. 由于各个 RTI 的结构不同, 参与测试的软件在各个主机上的部署方式也略有不同, 分别为:

BH RTI 2.3: 服务主机启动 BH_TMServer, 各仿真主机启动 BH_RtiExec 和 Benchmark;

DMSO RTI1.3NG-V6: 服务主机启动 DMSO_RTI, 各仿真主机启动 Benchmark;

Pitch pRTI1516LE: 服务主机启动 pRTI, 各仿真主机启动 Benchmark;

Mak RTI 3.0: 服务主机启动 Mak RTI, 各仿真主机启动 Benchmark.

4.2 性能指标

时间推进速度(time advance speed, TAS)和空载网络带宽占用量(vacant network bandwidth occupied, VNBO)是时间同步算法的常规性能指标. 引入 IP 组播是本文方法的一个重要特点, 为了验证其可行性, 以丢包影响率(loss effect ratio, LER)作为衡量该时间同步算法的一项性能指标. 此外, 为了考察仿真规模的扩展性, 用仿真成员规模(simulation member scale, SMS)进行衡量.

时间推进速度是指仿真节点在单位时间内的时间推进次数, 用于衡量节点进行一次时间同步协调的效率, 单位为次/秒. 设在t_1到t_2时间段进行了n_timeAdvance次时间推进, 则

$$\mathrm{TAS} = \mathrm{n\_timeAdvance}/(t_2-t_1).$$

空载网络带宽占用量是指仿真运行过程中, 在没有事件报文的情形下, 单位时间内RTI时间推进所占用的网络带宽量, 用于衡量主机节点和时间同步的网络带宽, 单位为Mbps. 测试中采用带宽分析软件获得数据. 也可以这样大致计算, 在t_1到t_2时间段内, 设主机通信的报文共有n个, $\{m_1, m_2, m_3,\ldots,m_n\}$, 每个报文的长度分别为$\{l_1, l_2, l_3,\ldots,l_n\}$, 则

$$\mathrm{VNBO} = \sum_{i=1}^{n}(m_i * l_i)/(t_2 - t_1).$$

丢包影响率是指组播发生丢包时节点的推进速度与没有发生丢包时节点推进速度的比值, 用于衡量组播丢包率对时间管理的影响程度, 单位为百分比. 设组播丢包率为 x%时的节点时间推进速度为 tas_lostX, 组播丢包率为 0%(即不发生丢包)时节点的时间推进速度为 tas_noLoss, 则 LER 可以表示为

$$\mathrm{LER} = \mathrm{tas\_lostX} / \mathrm{tas\_noLoss}.$$

时间同步成员规模是指仿真成员进行时间同步推进时, 成员数量与系统推进速度的关系, 用于测试 RTI 时间管理对仿真成员规模的支持.

4.3 实验结果分析

时间推进速度的实验结果如图 7 所示. 对于 BH RTI, 其中一台启动 TMServer, 其余 2 台(5/10 台)启动 RtiExec 和不同数量的 Benchmark 成员程序. 对于其他 3 种 RTI, 其中一台启动 RTI, 其余 2 台(5/10 台)启动不同数量的 Benchmark 成员程序, 成员的 Lookahead 取值为 1, 时间推进步长取值为 10, 分别测试了在不同数量主机以及不同数量仿真节点的多种情形下的时间推进速度. 在图 7(a)所示实验规模下, BH RTI 的时间推进速度高于其他 3 种 RTI. 图 7(b)实验中, 随着仿真成员数量的增多, 各个 RTI 的时间推进速度较 2 个仿真主机规模时均有下降, 其中实验所用版本的 Mak RTI 最多只允许加入 37 个成员, BH RTI 的时间推进速度仍高于其他 3 种 RTI. 图 7(c)实验中, 相对规模较大, 同样, 随着节点规模的增大, 各 RTI 时间推进速度降低. 其中, Mak RTI 不支持 50 或 100 个成员的规模, DMSO RTI 不支持 100 个成员的规模. 相对

于 5 个主机规模, BH RTI 下降的幅度较小, 并且可以在 100 个成员情形时, 仍保持较高的时间推进速度.

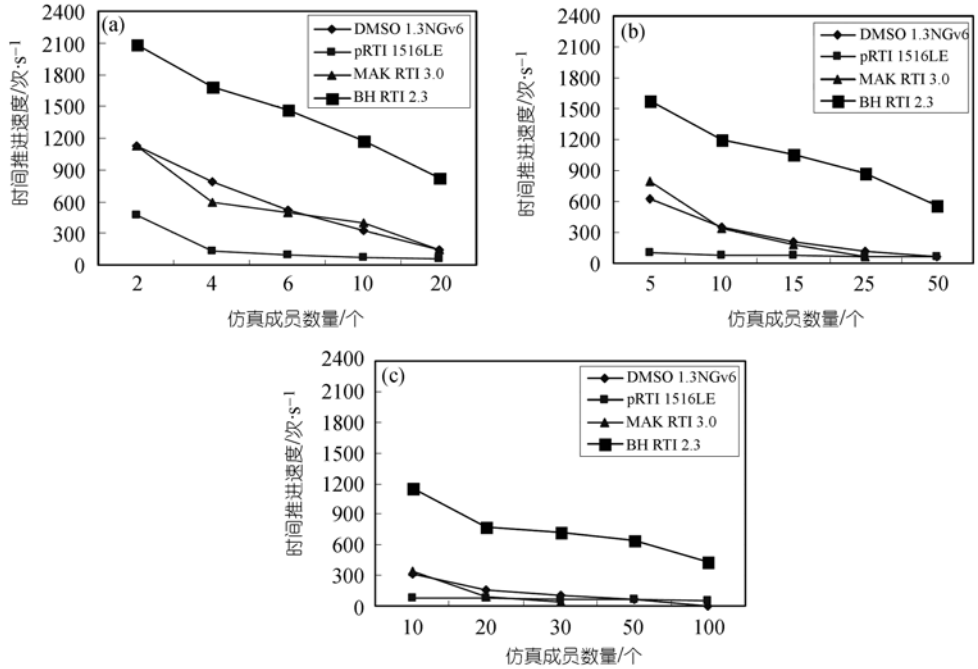

图 7 时间推进速度测试结果

(a) 2 台仿真主机的 TAS; (b) 5 台仿真主机的 TAS; (c) 10 台仿真主机的 TAS

在图 7 时间推进速度的实验过程中, RTI 时间管理服务运行时的网络带宽占用量如图 8 所示. 实验使用网络带宽测量软件 IRIS v4.07.1, 由于所用的 Pitch pRTI 为限制版, 一次仿真运行只能调用服务接口 100 次, 无法对其进行准确测量, 因此该组实验中不包含 pRTI. 图 8(a)为 2 台仿真主机规模下的实验结果, Mak RTI 带宽占用量相对较高, 最高时约占总带宽的 28%, 其次是 DMSO RTI, BH RTI 相对较低. 图 8(b)为 5 台仿真主机规模情形下的实验结果, 由于 Mak RTI 的成员数量限制, 因此没有对其进行节点规模为 50 的实验. 同样, 占用量最高的还是 Mak RTI, 次高的为 DMSO RTI, BH RTI 最低. 图 8(c)为 10 台仿真主机规模情形下的实验结果, 其中 DMSO RTI 和 Mak RTI 均不支持成员数量为 100 的仿真规模. 在 10 个仿真主机, 100 个成员节点的情形下, BH RTI 时间管理服务的网络带宽占用量约为总带宽的 3%.

在进行带宽测量的过程中, BH RTI 2.3 时间管理服务器节点的空载网络带宽的占用情形如图 9 所示. 可以看出, 作为协调中心的专用服务器, 其空载带宽占用量也只比 RTI 略高.

仿真成员分别为 20, 50, 100 的仿真规模下的丢包影响率实验结果如图 10 所示, 仿真主机数量分别为 2, 5, 10, 每台主机加入 10 个仿真成员. 该部分实验通过在 RTI 接收底层对组播报文进行丢弃的方法来模拟平均丢包率分别为 0~100%的 11 种组播丢包情形. 仿真成员的 Look-ahead 取值为仿真主机号, 时间推进步长统一取值为 1. 实验结果表明, 随着仿真规模的增大,

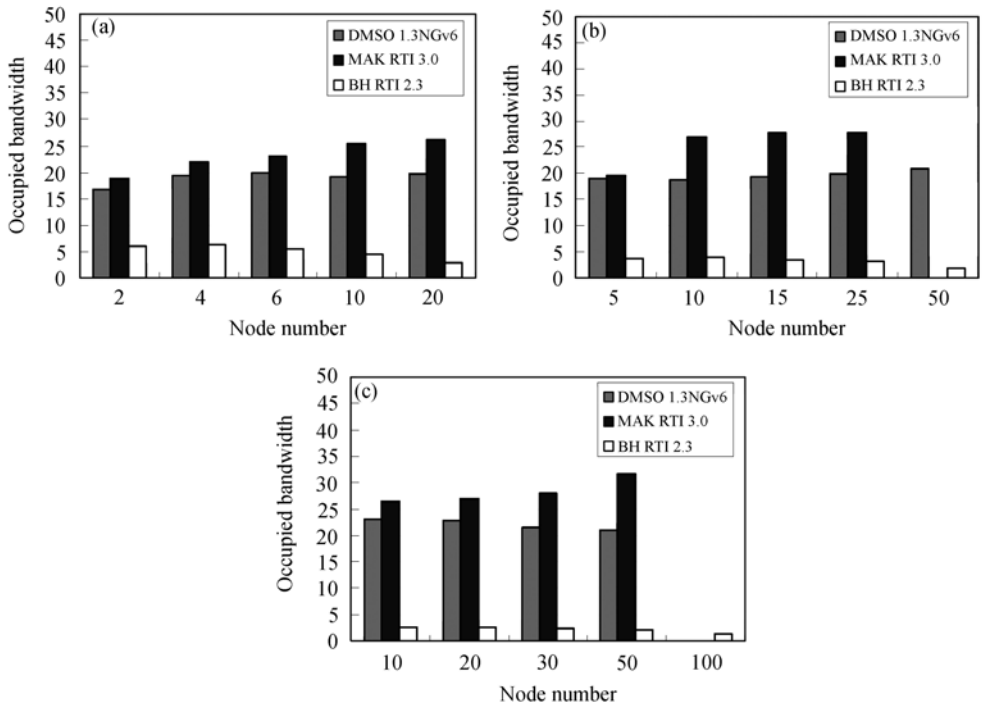

图 8　网络带宽占用量测试结果

(a) 2 台仿真主机的 VNBO; (b) 5 台仿真主机的 VNBO; (c) 10 台仿真主机的 VNBO

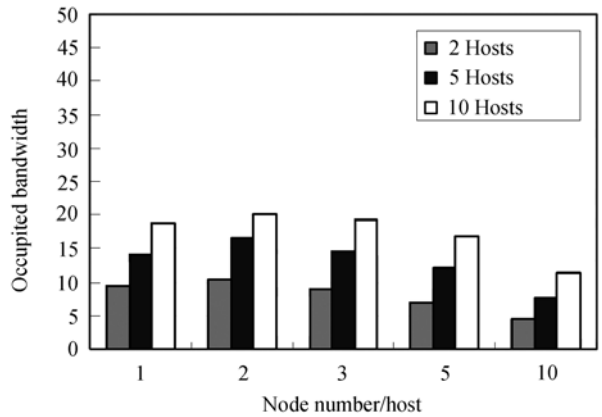

图 9　时间管理服务器的空载网络带宽占用量测试结果

组播丢包率对仿真成员的时间推进速度影响也随之增大. 存在丢包时, 仿真成员仍能正确推进. 在比较极端的情形下, 如组播丢包为 100%时, 3 种规模下的仿真成员推进速度分别不发生丢包时的 79%, 76%和 57%, 不会出现仿真无法进行时间推进的情形, 只是时间推进速度有所下降.

BH RTI 2.3 时间管理的仿真成员规模测试如图 11 所示. 实验在仿真主机规模为 10 台, 仿

真成员数量为 100~1000 时, 测试了各个成员的时间推进速度. 从图中可以看出, 随着仿真成员数量的增多, 节点的时间推进速度下降, 并逐渐趋于平缓. 当仿真成员数量为 1000 时, 节点的时间推进速度大概在 50 次/s 左右, 与 pRTI 1516LE 在 100 个成员时的时间推进速度相近, 与 DMSO RTI 1.3NGv6 在 50 个成员时的时间推进速度相近, 略高于 MAK RTI 3.0 在 30 个仿真成员规模下的时间推进速度. 实验数据表明, BH RTI 2.3 的时间管理可以支持较大规模的仿真, 1000 个盟员的时间同步推进速度可达到约 50 次/s.

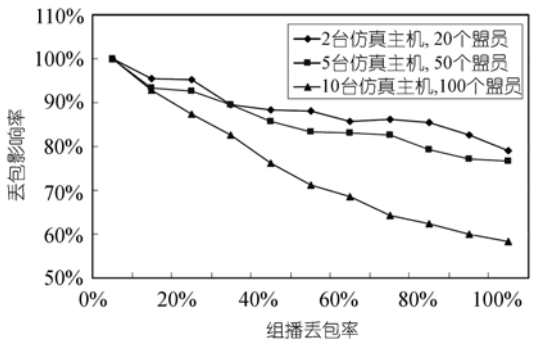

图 10 丢包影响率测试结果

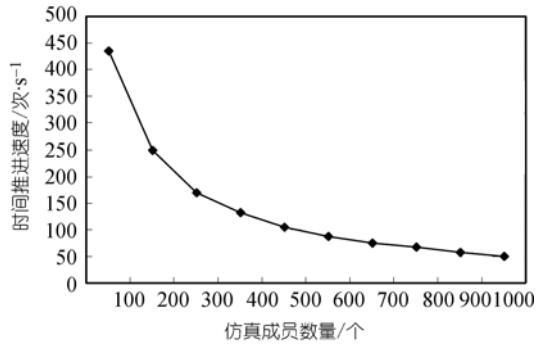

图 11 仿真成员规模测试结果

5 结束语

本文研究了分布式仿真中的时间同步算法, 提出一种可用于大规模分布节点的仿真时间同步算法. 通过对分布式仿真中节点时间推进上限以及节点协调过程特点进行分析, 给出了分组管理的 LBTS 计算模型以并提出了时间同步控制报文可靠性定理. 根据节点与控制报文之间的组通讯特点及各个节点对不同报文可靠度要求的不同, 引入了 IP 组播来处理一些控制报文的组通信, 结合可靠单播实现控制报文的传输. 最后, 将本文的算法应用于 BH RTI 时间管理服务进行实验和验证, 并与同类系统的时间管理算法进行了比较. 实验数据表明, 该算法在仿真成员的时间推进速度、网络带宽占用量等方面均优于对比实验中的其他系统的时间管理算法, 具有良好的规模扩展性, 1000 个盟员的时间同步推进速度可达到约 50 次/s, 相当于同类系统中 100 个节点时的推进速度.

参考文献

1 Fujimoto R M. Parallel simulation: parallel and distributed simulation systems. In: Proceedings of the 33th Winter Simulation Conference. Virginia: IEEE, 2001. 147—157
2 Fujimoto R M. Parallel simulation: distributed simulation systems. In: Proceedings of the 35th Winter Simulation Conference. Louisiana: ACM, 2003. 124—134
3 Fujimoto R M. Time management in the high level architecture. Simulation, 1996, 71(6): 60—67
4 Cai W T, Turner S J, Lee B S, et al. An alternative time management mechanism for distributed simulations. ACM Transactions on Modeling and Computer Simulation, 2005, 15(2): 109—137 [DOI]

5 Chandy K M, Misra J. Distributed deadlock detection. ACM Trans Comput Syst 1983, 1(2): 4—9
6 Mattern F. Efficient algorithms for distributed snapshots and global virtual time approximation. J Parall Distr Comput, 1993, 18(4): 423—424 [DOI]
7 Fujimoto R M. Exploiting temporal uncertainty in parallel and distributed simulation. In: Proceedings of the 13th Workshop on PADS. Georgina: IEEE, 1999. 46—53
8 Lee B S, Cai W T, Zhou J L. A causality based time management mechanism for federated simulation. In: Proceedings of the 15th IEEE/ACM/SCS Workshop on PADS. California: IEEE, 2001. 83—90
9 Wang X G, Turner S J, Low M Y H, et al. Optimistic synchronization in HLA-based distributed simulation. In: Proceedings of the 18th IEEE/ACM/SCS Workshop on PADS. Austria: ACM, 2004. 123—130
10 Morillo P, Orduna J M, Duato J. A scalable synchronization technique for distributed virtual environments based on networked-server architectures. In: Proceedings of the 2006 International Conference on Parallel Proceeding. Ohio: IEEE, 2006. 74—81
11 Fujimoto R M, Weatherly R M. Time Management in the DoD High Level Architecture. In: Proceedings of the 10th Workshop on PADS. Pennsylvania: IEEE, 1996. 60—67
12 Carothers C D, Weatherly R M, Fujimoto R M, et al. Design and implementation of HLA time management in the RTI version F.0. In: Proceedings of the 29th Winter Simulation Conference. Georgia: IEEE, 1997. 373—380