HLA/RTI 时间管理的一种优化算法

姚益平,卢锡城
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 时间管理服务是 HLA 与以前分布式仿真标准最主要区别内容之一,也是 RTI 实现的重点和难点.其性能直接影响到仿真的效率和正确性.时间管理服务实现的关键是最大可用逻辑时间 GALT 的计算.论文针对目前 GALT 算法存在的可能死锁问题,提出了计算 GALT 的递归式算法——R-GALT 算法和递归式推进检测方法,并给出了 R-GALT 算法无死锁证明.R-GALT 算法不但解决了时间管理实现中可能出现的联盟时间推进死锁问题,而且能够提高盟员时间推进的效率.该算法在作者等人研制的遵循 IEEE1516 标准的 RTI 软件 StarLink 中已经得到了实现,测试表明,其性能优于国际上同类软件.

关键词: 高层体系结构 (HLA); 运行支持环境 (RTI); 时间管理 (TM); 最大可用逻辑时间 (GALT)
中图分类号: TP391.9 **文献标识码**: A **文章编号**: 0372-2112 (2004) 02-0294-04

An Optimized Algorithm of HLA/RTI Time Management

YAO Yi-ping, LU Xi-cheng
(Computer College of National University of Defense Technology, Changsha, Hunan 410073, China)

Abstract: Time management service is one of the main features which distinguish HLA from previous distributed interactive simulation standards, and is the most important and difficult part in RTI implementation. Its implementation method can affect the efficiency and validity of simulation directly. The key problem of the implementation of time management service is the computation of Greatest Available Logical Time (GALT). Aiming at the possible deadlock problem of the current GALT algorithm, an optimized algorithm called recursive GALT (R-GALT) algorithm along with the idea of recursive advancing check is proposed. A formalized verification is also presented. Compared with the original algorithm, the new algorithm can not only enhance the efficiency, but also avoid deadlock. The algorithm has been used in the design of RTI software StarLink which is based on CORBA and in accord with IEEE 1516. The measurement shows that the performance of time management of StarLink is better than of pRTI1516 (it's supposed to be the best RTI product in the world).

Key words: high level architecture; runtime infrastructure; time management; greatest available logical time

1 引言

高层体系结构 HLA(High Level Architecture)为分布式交互仿真提供了一个通用的技术框架和开放的标准.HLA 已于 2000 年 9 月被定为国际分布仿真通用标准 IEEE1516.运行支撑环境 RTI(RunTime Infrastructure) 是 HLA 的具体实现.时间管理服务是 HLA 区别于以前分布式仿真标准的一项重要服务,其目的是控制各盟员在仿真时间轴上的推进,保证 RTI 能在适当的时间以适当的方式和顺序将来自盟员的消息转发给相应的盟员.时间管理服务实现的主要内容是时间推进请求服务的实现,而时间推进请求服务实现的关键是推进时机的确定[3].在 IEEE1516 中,为了保证受限盟员不会接收到比它的当前逻辑时间值小的时戳消息,定义了一个被称为最大可用逻辑时间 GALT(Greatest Available Logical Time)的术语来限制盟员的推进[1].当盟员调用 TAR(Time Advance Request)或 NMR(Next Message Request)请求时,若请求前进的逻辑时间 ≥ GALT,请求将被挂起,直到界限的增长超过请求的逻辑时间;当盟员调用 TARA(Time Advance Request Available)或 NMRA(Next Message Request Available)请求时,仅当请求前进的逻辑时间 > GALT 时,请求被挂起.由此可以看出,GALT 的计算是时间管理算法的核心内容.高效、无死锁的 GALT 算法一直是研究的热点和难点.

2 目前的时间管理算法及其存在的问题

为了便于 GALT 的计算,Frederick 等人引入了盟员输出时间 OPT(OutPutTime),用来表示该盟员现在及将来发出的 TSO(Time Stamp Order)消息中所能带的最小时戳值[2].因为只有校准盟员能够发送 TSO 消息,所以也只有校准盟员才被考虑

收稿日期:2003-03-18;修回日期:2003-11-10
基金项目:国家 863 计划项目(No.2001AA115127);国家自然科学基金(No.60373024);武汉大学软件工程国家重点实验室开放研究基金(No.SKL(4)005)

计算它们的 OPT,因此校准盟员的时间推进非常关键,它可能引发联盟中相关受限盟员的时间推进. 盟员的 OPT 取决于盟员的当前状态、当前逻辑时间值或请求推进逻辑时间、前瞻值 Lookahead 和 TSO 队列中消息时戳的最小值. OPT 计算公式如下:

(1) 若盟员是非校准的,由于其时间推进对其他盟员时间推进不会产生影响,所以无需计算它的 OPT.

(2) 若盟员为校准盟员(以下同),当该盟员处于时间准许状态时:

$$OPT(i) = T_C(i) + T_L(i) \quad (1)$$

其中 $T_C(i)$ 为盟员的当前逻辑时间,$T_L(i)$ 为该盟员前瞻值 lookahead.

(3) 当盟员处在时间推进状态的 TAR/TARA 请求等待时:

$$OPT(i) = T_R(i) + T_L(i) \quad (2)$$

其中 $T_R(i)$ 为盟员请求推进的逻辑时间,$T_L(i)$ 同前.

(4) 当盟员处在时间推进状态的 NMR/NMRA 请求等待时:

$$OPT(i) = \min\{T_R(i) + T_L(i), minTSO(i), GALT(i)\} \quad (3)$$

其中 $T_R(i)$、$T_L(i)$ 同前,$minTSO(i)$ 为已经存在于 RTI 所维护的盟员 i 的 TSO 消息队列中消息带有的最小时戳值,$GALT(i)$ 为盟员 i 的 GALT 值,min 表示求最小值.

Frederick 将盟员 GALT 的值定义为所有其他校准盟员输出时间的最小值,即

$$GALT(i) = \min\{OPT(j)\} \quad (4)$$

其中 $j \neq i$,且盟员 j 为校准盟员.

上述算法较好地解决了盟员请求时间推进时的 GALT 计算问题,但是在实现该算法的过程中,我们发现,该算法可能引起死锁.

假设在一个只有两个盟员的联盟执行中,两盟员都为校准(Time Regulating,TR)且受限(Time Constrained,TC)盟员,它们相互订购了对方的时戳消息,因此在时间推进时相互约束.

设两盟员的当前逻辑时间分别为 T_{C1} 和 T_{C2}(见图 1). 考虑联盟以下的时间推进过程:盟员 1 提出 NMR(T_{R1}) 调用,请求前进到时间 T_{R1},根据式(4)、(1),可知:

$$GALT(1) = OPT(2) = T_{C2} + T_{L2}$$

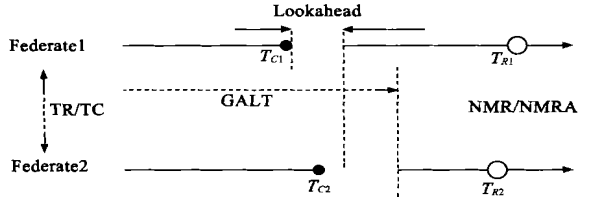

图 1　具有两盟员的联盟死锁示例

由 GALT 对受限盟员时间推进请求的限制可知,当 $T_{R1} > GALT(1)$ 并且盟员 1 的 TSO 消息队列中无消息或 $minTSO(1) > GALT(1)$ 时,推进请求不会被准许,直到 GALT 的值增加到大于 T_{R1} 或大于 $minTSO(1)$. 因此,盟员 1 处在 NMR 请求挂起状态. 随后,盟员 2 请求时间推进到 T_{R2}(设 $T_{R2} > T_{C2} + T_{L2}$,且盟员 2 的 TSO 消息队列中无消息或 $minTSO(2) > T_{R2}$),根据式(4)、(3)可知:

$$GALT(2) = OPT(1) = \min\{T_{R1} + T_{L1}, minTSO(1), GALT(1)\} \quad (5)$$

从上述公式中可以看出,要确定 GALT(2) 的值,关键是确定 GALT(1) 的值,而 GALT(1) 的计算有两种方法,一是采用原有的值,这时由假设 $minTSO(1) > GALT(1)$ 及 $T_{R1} + T_{L1} > T_{R1} > GALT(1)$,可知:$GALT(2) = GALT(1) = T_{C2} + T_{L2} < T_{R2}$. 因此,盟员 2 的推进请求也不被准许,进入请求挂起状态. 这样,两个盟员都进入了请求挂起状态,每个盟员都在等待其他盟员的时间推进,联盟出现死锁. 第二种方法是重新计算 GALT(1) 的值,此时根据式(4)、(3)可知:

$$GALT(1) = OPT(2) = \min\{T_{R2} + T_{L2}, minTSO(2), GALT(2)\} \quad (6)$$

从式(5)、(6)可以看出,GALT(2) 的计算依赖于 GALT(1) 的值,而 GALT(1) 的计算又依赖于 GALT(2) 的值,出现了死循环.

解决上述问题有两种经典的途经,一是对算法进行优化,避免死锁或死循环的发生,二是在运行过程中对死锁进行检测,当发生死锁时,采用一定的规则将死锁打破[4].

3　递归式算法的提出

在引入递归式算法之前,我们先对式(4)做一个优化,由于盟员的前进只与对它有约束关系的盟员有关,因此我们规定,式(4)中的 j 为所有对盟员 i 时间推进有约束的盟员(而不是所有除 i 之外的校准盟员),为了不影响正确性,我们还规定,中途加入对 i 约束的校准盟员,其逻辑时间与前瞻值的和不能小于盟员 i 的逻辑时间. 显然,这样规定既不影响正确性,同时又缩小了计算盟员 i 的 GALT 时要考虑的盟员的范围,减少了 OPT 的计算量,提高了系统的效率.

不难看出,上述算法死锁的原因主要由于当盟员 i 处在 NMR/NMRA 请求等待时,它的输出时间 $OPT(i)$ 不确定造成的. 这种不确定使得处于 NMR/NMRA 请求等待、并且两两相互约束的盟员互相等待而出现死锁(见图 2).

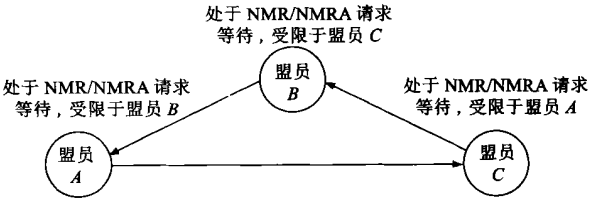

图 2　死锁图例

为此,我们对式(3)进行改进,即在式(3)中引入递归机制,通过递归找出影响盟员输出时间的所有可能因素,从而求出盟员确定的输出时间,经改进后的公式如下:

$$OPT(i) = \min\{T_R(i) + T_L(i), minTSO(i), \min\{OPT(j)\}\} \quad (7)$$

其中 j 为所有对盟员 i 时间推进有约束的盟员.

该公式是一个递归公式,通过盟员间约束关系的传递,可以找到一个有约束关系的盟员集. 为避免死循环,我们规定每个盟员的 OPT 只在式(7)中最多计算一次,当某个盟员 k 的

OPT(k)在式(7)中第二次出现时,直接将它从公式中去掉.

4 递归式算法 R-GALT 的描述

由式(4)可知,一个盟员的 GALT 值的计算关键是所有对该盟员时间推进有约束的校准盟员的输出时间 OPT 的计算.

当需要计算一个处于 NMR/NMRA 请求挂起状态的校准盟员的 OPT 时,我们从该盟员开始采用深度优先的递归算法,对所有对该盟员有约束关系(包括直接约束和间接约束)的校准盟员进行遍历,若搜索到的盟员 a 处于时间准许状态或 TAR/TARA 请求等待状态,则返回相应的 OPT 值,回溯. 若盟员 a 属于第二次被搜索到,则返回一个系统所能表示的最大值(设为 MaxValue),回溯. 若盟员 a 处于 NMR/NMRA 请求等待状态,则继续对所有对盟员 a 有约束关系的盟员进行递归.

根据式(1)、(2)、(7),我们有计算盟员 i 的 GALT 值的递归式算法——R-GALT 算法如下:

```
(1) OPT(j)
{
  if(盟员 j 属于第二次被搜索到)  then
    return(MaxValue);
  else if(盟员 j 处于时间准许状态)  then
    return(TC(j) + TL(j));
  else if(盟员 j 处于 TAR/TARA 请求等待状态)  then
    return(TR(j) + TL(j));
  else /*(盟员 j 处于 NMR/NMRA 请求等待状态)*/
  {设所有对盟员 j 时间推进有直接约束的盟员为 j1、j2、…、js;
    令 m = min{TR(j) + TL(j), minTSO(j)};
    for(k = 1; k ≤ s; k++)
        m = min{m, OPT(jk)};
    return(m);
  }
}
```

(2) 设所有对盟员 i 时间推进有直接约束的校准盟员为 i_1、i_2、…、i_n;

```
GALT = MaxValue;
for(k = 1; k ≤ n; k++)
GALT = min{GALT, OPT(ik)};
```

(3) 结束.

5 R-GALT 算法无死锁证明

由于处于 TAR/TARA 请求等待或处于时间准许状态的盟员的输出时间是确定的,因此,这些盟员不会引起死锁或死循环. 于是,要证明 R-GALT 算法不会引起死锁或死循环,我们只需证明处于 NMR/NMRA 请求等待、并且两两相互约束的盟员集合中存在能够推进的盟员即可.

证明 设 A = 盟员 i 及所有对该盟员有约束关系(包括直接约束和间接约束)的盟员集合;

$S = A$ 中处于 NMR/NMRA 请求等待、并且两两相互约束的盟员集合. 下面分两种情况来证明:

(1) 设盟员 $k \in S$,且 minTSO(k) 是 S 中所有盟员的 $T_R + T_L$ 和 minTSO 中的最小值,则 $T_R(k) + T_L(k) \geq$ minTSO(k). 不失一般性,我们假设 S 之外对盟员 k 有约束的盟员是可以推进的,这样,这些盟员的 OPT 在某个时刻之后一定可以 $\geq$ minTSO(k);又根据前面 minTSO(k) 是 S 中所有盟员的 $T_R + T_L$ 和 minTSO 中的最小值的假设及式(7),我们有 S 中除 k 之外的所有盟员的 OPT $\geq$ minTSO(k);因此,根据式(4)有:GALT(k) $\geq$ minTSO(k),此时有两种情况:

①GALT(k) > minTSO(k):

●若 $T_R(k)$ > minTSO(k),则盟员 k 可推进到 minTSO(k);

●若 $T_R(k) \leq$ minTSO(k),则盟员 k 可推进到 $T_R(k)$.

②GALT(k) = minTSO(k):

●若 $T_R(k)$ > minTSO(k),则 RTI 所维护的盟员 k 的 TSO 消息队列中时戳为 minTSO(k) 的消息可发送出去,完了之后当重新计算 GALT(k) 时,新的 GALT(k) 值一定大于原来的 minTSO(k),因此,盟员 k 可推进到 minTSO(k);

●若 $T_R(k)$ < minTSO(k),则盟员 k 可推进到 $T_R(k)$;

●若 $T_R(k)$ = minTSO(k) = GALT(k),则 RTI 所维护的盟员 k 的 TSO 消息队列中时戳为 minTSO(k) 的消息可发送出去,完了之后重新计算 GALT(k) 时,根据式(7):当 $T_L(k) \neq 0$ 时,新的 GALT(k) 值一定大于 $T_R(k)$,因此,盟员 k 可推进到 $T_R(k)$;当 $T_L(k) = 0$ 时,新的 GALT(k) 值仍等于 $T_R(k)$,此时采用 NMRA,盟员可推进到 $T_R(k)$.

(2) 设盟员 $k \in S$,且 $T_R(k) + T_L(k)$ 是 S 中所有盟员的 $T_R + T_L$ 和 minTSO 中的最小值,则 minTSO(k) $\geq T_R(k) + T_L(k) \geq T_R(k)$. 同样,我们可假设 S 之外对盟员 k 有约束的盟员是可以推进的,这样,这些盟员的 OPT 在某个时刻之后一定可以 $\geq T_R(k) + T_L(k)$;又根据前面 $T_R(k) + T_L(k)$ 是 S 中所有盟员的 $T_R + T_L$ 和 minTSO 中的最小值的假设及式(7),我们有 S 中除 k 之外的所有盟员的 OPT $\geq T_R(k) + T_L(k)$;因此,根据式(4)有:GALT(k) $\geq T_R(k) + T_L(k) \geq T_R(k)$,此时设 $T_L(k) \neq 0$,则 GALT(k) > $T_R(k)$,盟员 k 可推进到 $T_R(k)$;设 $T_L(k) = 0$,则若 GALT(k) > $T_R(k)$,盟员 k 可推进到 $T_R(k)$;若 GALT(k) = $T_R(k)$,此时采用 NMRA,盟员 k 可推进到 $T_R(k)$.

综上所述,R-GALT 算法不会引起死锁或死循环.

6 推进检测的提出

为避免不必要的等待,当一个盟员请求时间推进时,需要在一个该请求前进可影响的盟员集合中考虑这次前进是否可以推动其他处在挂起状态的盟员得以前进. 因此我们提出了盟员时间推进检测的思想,即当盟员 i 请求时间推进时,在计算盟员 i 的 GALT 值及判断它能否推进的同时,还要进行盟员 i 推进影响的检测,查看该盟员提出时间推进请求后,它所约束的盟员(包括直接约束和间接约束)此时可否因为盟员 i 的这次请求而从挂起状态转变为时间准许状态. 即需要重新计算它所约束的盟员的 GALT,过程与前述类似.

7 结束语

GALT 的计算是时间管理实现的关键,本文对 Frederick 等人提出的算法进行了改进,经改进后的算法与原算法相比,不

仅提高了效率,而且避免了死锁.该算法在我们研制的遵循 IEEE1516 标准的 RTI 软件 StarLink[5]中已经得到了实现.经与国际上同类软件 pRTI1516 进行对比测试表明(见图 3),其时间推进请求延迟小于国际上同类软件.

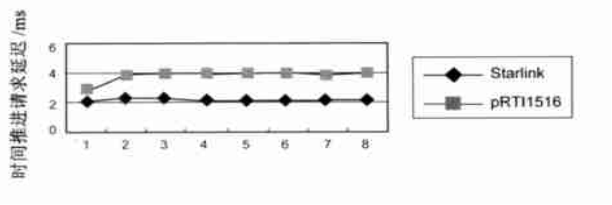

图 3　时间推进请求延迟测试结果

致谢　在本算法的研究探讨过程中与王怀民教授、韩林同志及刘步权博士生进行了多次有益的讨论,并获得了一些好的建议.该算法提出后由韩林同志进行了具体编程实现和验证.在此一并表示衷心的感谢!

参考文献:

[1] Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - - IEEE Std 1516-2000, 1516.1-2000, 1516.2-2000 [S]. New York: The Institute of Electrical and Electronics Engineers Inc, 2000.

[2] Frederick Kuhl, Richard Weatherly, Judith Dahmann. Creating Computer Simulation Systems[M]. Prentice Hall PTR. Upper Saddle River, NJ 074584, 1999.

[3] R M Fujimoto. Time management in the high level architecture [J]. Simulation, 1998, 71(6): 388 - 400.

[4] 张龙,尹文君,等. RTI 系统时间管理算法研究[J]. 系统仿真学报, 2000, 12(5): 494 - 498.

[5] 姚益平,卢锡城,王怀民. 层次式 RTI 服务器的设计与实现[J]. 计算机学报, 2003, 26(6): 716 - 721.

作者简介:

姚益平　男,1963 年生于湖南邵东,硕士,教授,主要研究方向为分布式仿真、软件工程和虚拟现实等. Email: ypyao@nudt.edu.cn.

卢锡城　男,1946 年 11 月生于江苏靖江,国防科技大学副校长、中国工程院院士、博士生导师,主要研究领域为高性能计算机和高性能网络技术.