软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(8):2188−2206 [doi: 10.3724/SP.J.1001.2012.04113]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

多核环境下面向仿真组件的 HLA 成员并行框架*

彭 勇+, 蔡 楹, 钟荣华, 黄柯棣

(国防科学技术大学 机电工程与自动化学院,湖南 长沙 410073)

Parallel Framework for HLA Federate Oriented to Simulation Component on Multicore Platform

PENG Yong+, CAI Ying, ZHONG Rong-Hua, HUANG Ke-Di

(College of Mechatronics Engineering and Automation, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: yongpeng@nudt.edu.cn, http://www.nudt.edu.cn

Peng Y, Cai Y, Zhong RH, Huang KD. Parallel framework for HLA federate oriented to simulation component on multicore platform. *Journal of Software*, 2012,23(8):2188−2206 (in Chinese). http://www.jos.org.cn/1000-9825/4113.htm

Abstract: A simulation component, oriented as a parallel framework for federate, was raised to facilitate federate development and improve execution performance of federate on multicore platform. Simulation components were employed to compose and assemble federates in parallel framework. With simulation engine management service, data distribution management service, object management service, component management service and load balancing of parallel framework, a multicore parallel environment was provided for simulation component, which also assured correct interactions between parallel federates and RTI. Experiments were carried out on the extra overhead introduced by parallel framework and performance comparisons between normal federate and parallel federate were made. Results showed that parallel framework fully exploited multicore processors with reduced execution time and improved performance of simulation system.

Key words: modeling simulation; HLA (high level architecture); parallel federate; simulation component; multicore

摘 要: 提出了一种面向仿真组件的并行联邦成员框架,以解决基于 HLA(high level architecture)复杂仿真系统联邦成员开发的问题,并提升多核处理器环境下联邦成员的运行性能.并行联邦成员框架通过仿真组件的组合、装配来构建联邦成员.通过仿真引擎管理、数据分发管理、对象管理、组件管理服务和负载平衡功能,并行联邦成员框架为仿真模型构建了一个多核的并行执行环境,并确保并行成员能与 RTI 正确交互.通过实验来研究并行成员框架引入的额外开销,并比较并行成员和普通成员的性能.实验结果表明,并行框架能够充分利用多核处理器的计算能力来减少仿真系统运行时间,提高系统性能.

关键词: 建模仿真;高层体系结构;并行成员;仿真组件;多核

中图法分类号: TP316 **文献标识码**: A

* 基金项目: 国家自然科学基金(61074108)
收稿时间: 2011-04-13; 修改时间: 2011-07-01; 定稿时间: 2011-08-09

© 中国科学院软件研究所 http://www.jos.org.cn

建模仿真是对复杂系统进行实验分析的一种有效手段.基于 HLA(high level architecture)的大规模复杂仿真在科学研究和工程实验领域都有非常广泛的应用.复杂仿真系统包含成千上万的元素或子系统,元素之间的关系更加错综复杂.复杂仿真系统的开发是一个耗时、复杂的系统工程,模型重用能够加快系统的开发过程,节省开发成本,提高模型的质量.发展 HLA 标准的目的之一是促进仿真重用[1].一方面,基于组件的软件工程(component based software engineering,简称 CBSE)方法是实现软件重用的重要方法.用组件的方式封装对象类和交互类的功能,构建仿真组件.在组件管理框架的支持下,通过组件组合和装配,可以按照需求构建具有不同功能的联邦成员.与联邦成员相比,组件的功能内聚,可以实现灵活、快速的组合,具有更高的可重用性.联邦成员的组件化有利于模型的组合、重用,同时也是实现联邦成员并行化的一种方法.另一方面,计算机软硬件的不断发展以及对计算速度的需求,也对 HLA 的研究也有深远的影响.特别是多核处理器的普及应用迫切需要改变联邦成员的开发方式,以便能够充分利用多核处理器的计算能力,提升仿真系统的性能.随着对处理器性能需求的不断提高,处理器的发展已经从单纯提高 CPU 的频率转变为发展多核模式[2].在 HLA 标准中,传统成员是串行处理的.在多核 CPU 的情况下,传统联邦成员已经不能充分利用多核 CPU 的计算资源,成为仿真系统性能的瓶颈.联邦成员的并行化是提高仿真系统运行效率的一种有效方法.

为了解决仿真模型重用和联邦成员并行化的问题,本文提出一种基于组件的并行联邦成员框架.首先,并行联邦成员框架提供了和 HLA 标准兼容的仿真组件模型,支持以仿真组件方式构建联邦成员.其次,并行联邦成员框架实现了仿真引擎管理服务,数据分发管理服务,对象管理服务、组件管理服务以及成员内部的动态负载平衡功能.这些服务为仿真实体并行运行提供了一个支持平台.仿真引擎管理服务提供基于线程的逻辑进程驱动仿真模型运算,并实现了改进的集中式栅障保守时间同步算法,以减少逻辑进程在时间同步的等待时间.并行成员框架通过 min-max exchange balancing 算法和逻辑进程之间仿真实体的动态迁移实现负载平衡.最后,并行联邦成员框架通过 RTI 适配器和 RTI 进行时间同步和通信,实现和其他联邦成员交互.

本文第 1 节是背景介绍和问题分析.第 2 节给出相关工作.第 3 节详细介绍基于组件的并行联邦成员框架的功能服务和仿真组件模型设计.第 4 节研究并行联邦成员框架负载平衡功能.第 5 节是实验设计和结果分析.第 6 节给出结论和下一步的工作.

1 背景和问题分析

高层体系结构是分布仿真领域的一个软件构架标准[1].HLA 标准由 3 部分组成:HLA 规则、对象模型规范和 RTI 接口规范.RTI 接口服务可以分为 6 类:联邦管理、声明管理、对象管理、时间管理、数据分发管理和所有权管理,如图 1 所示.联邦成员通过调用 RTI 的服务和实现 RTI 的回调函数和其他成员进行时间同步和交互.联邦成员可能包含大量的仿真实体(模型),并且作为一种特殊的应用程序联邦成员内部的仿真实体之间没有并行计算的能力,从而导致联邦成员的计算过程是一个非常耗时的过程,成为仿真系统性能的瓶颈.在多核、众核处理器平台下,研究并行联邦成员和实现联邦成员内仿真实体处理过程的并行化,以便充分利用处理器的计算资源来提高仿真系统的性能,是一个值得深入研究的课题.典型的联邦成员计算过程是:首先接收交互和属性更新,然后进行模型计算并更新属性和发送交互,最后请求时间推进.不断循环这个过程,直至仿真结束.在模型解算过程,联邦成员内仿真实体的计算函数是依次串行执行的,这是联邦成员不能充分利用多核处理器计算能力的根本原因.为了提高联邦成员的运行速度,可以对成员进行组件化改造.采用基于组件的方式实现成员并行化,不但要求仿真组件具有一般组件的特点,而且需要提供一个组件管理框架解决组件之间的数据分发管理、时间管理、组件组合、对象管理、负载平衡等问题,如图 1 的图例所示.联邦成员并行化涉及时间同步、数据分发和事件规划等问题,而基于组件的联邦成员开发则要考虑组件管理、仿真实体管理和组件组合等问题.这些都不是简单的问题.

© 中国科学院软件研究所 http://www.jos.org.cn

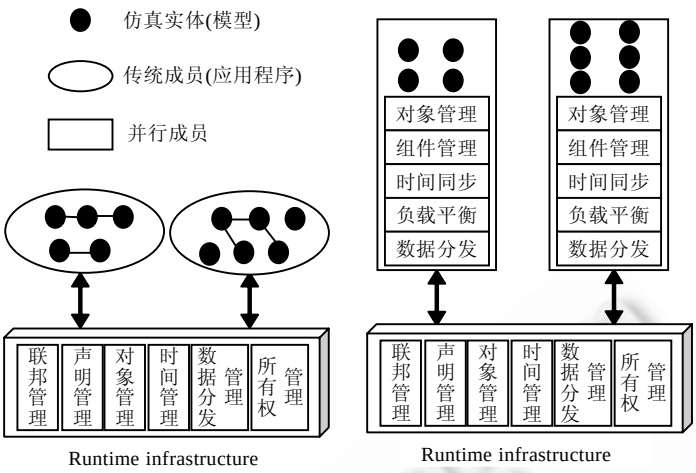


Fig.1 Structure of HLA simulation system based on traditional and parallel federates

图 1 基于传统成员和并行成员的 HLA 仿真系统结构

2 相关工作

为了限制讨论范围,这里我们不准备全面讨论基于组件的建模仿真问题,只关注面向 HLA 仿真系统的组件化建模仿真方法.

龚建兴等人研究了基于组件构建可扩展联邦成员的方法[3].Alex 等人借鉴基于组件开发(component-based development,简称 CBD)的思想研究了基于仿真组件的联邦成员开发方法[4].该方法将调用 RTI API 的代码从组件中分离出来,并改进了 RTI 的数据转换和传输服务;同时,通过组件集成框架提高基于 RTI 的组件和非基于 RTI 组件之间的聚合能力.在此基础上,Alex 改进了组件的聚合框架,提出了面向 HLA 的仿真组件模型(simulation component model,简称 SCM)[5].SCM 采用类似 CORBA 的标准组件模型,将仿真模型的逻辑过程包装成组件.组件聚合框架使用代码生成的方法将不同的组件集成为一个联邦成员.这种方法提高了联邦成员的互操作和可重用性,降低了开发联邦成员和联邦的难度.

Katarzyna 等人提出了一种 HLA 组件模型[6].这种模型综合应用 HLA 标准、组件技术和网格基础设施.按照 HLA 标准在网格的环境下实现 HLA 组件模型.该方法解决了网格环境下,使用仿真组件动态构建大规模复杂仿真系统的问题.

Benali 等人提出了一种基于组件的建模仿真方法[7].他们的组件模型由知识层、概念层和实现层构成,用元模型描述组件的领域知识.组件的概念模型采用基本对象模型(basic object model,简称 BOM)标准描述.实现层符合 HLA 标准.这种方法有利于复杂系统的建模和模型的集成.类似地,Farshad 等人也提出一种基于 BOM 的仿真模型[8,9].该仿真模型综合利用 BOM 概念模型、本体和仿真参考标记语言解决仿真模型的组合和重用问题,加快了仿真模型的开发和集成.Petty 等人对模型的可组合性进行深入的研究,研究了可组合的相关术语词典、组件选择算法的复杂度和组件的组合模式[10].

并行分布离散事件仿真的研究范围非常广泛,包括事件排序、内存管理、时间同步、消息传递等方面.时间同步算法是影响并行分布离散时间仿真系统性能的关键因素,因此相对于其他方面,时间同步算法得到了更加广泛和深入的研究.总的说来,并行离散事件仿真的时间同步算法可以分为两类:保守时间同步算法和乐观时间同步算法.在乐观时间同步算法中,Jefferson 提出的时间弯曲算法(time warp)最广为人知[11].关于时间同步算法的研究已经非常成熟,在这里不作详细讨论,各种算法的细节参见文献[12].目前,在众多的并行离散事件仿真平台中,SPEEDES 应用最多,最为成熟.它支持保守和乐观的时间管理机制[13].

Jeffrey 等人提出了一种标准仿真体系结构(standard simulation architecture)[14].标准仿真体系结构吸收了

© 中国科学院软件研究所 http://www.jos.org.cn

HLA 在时间管理、兴趣管理和声明管理的优点,同时结合了 SPEEDES 在高性能仿真方面的优势.标准仿真体系结构构建的系统称为 SSA 联邦.它是一种基于 SPEEDES 特殊的 HLA 成员.SSA 联邦通过自动的分布仿真管理服务(distributed simulation management services,简称 DSMS)为并行联邦内的实体提供类似 RTI 服务的功能,并利用 HLA 网关协调 DSMS 和 RTI 之间的交互.

HLA/RTI 作为分布仿真的标准体系结构,有很多实现,如国防科大的 KD-RTI[15]、StarRTI[16]、北航的 BH-RTI[17]、瑞典的 PitchRTI 以及 DMSO 的 RTI[18].其中,国防科大的 KD-RTI 在国内,特别是军用仿真领域应用最为广泛.这些 RTI 基本都是实现了保守的时间同步算法.此外,苏年乐等人研究了欧洲航天局的仿真模型可移植性规范 2(SMP2)仿真组件的并行化方法[19].

前面提到的基于组件建模仿真方法主要是解决联邦成员的开发和集成问题,并提高模型的可组合性和可重用性.这些方法并没有考虑仿真组件的并行化问题.

3 基于组件的并行联邦成员框架

3.1 概 述

并行组件框架可以分为 3 部分:仿真组件模型(simulation component model,简称 SCM)、PCOM(parallel multicore emulator for component)和 RTI 适配器(RTI adapter),如图 2 所示.图 2 中的虚线矩形表示并行组件框架.并行组件框架中,最上面的矩形包含的是 3 个仿真组件.仿真组件是实现了 SCM 标准接口的用户模型.SCM 的标准接口包括管理接口和运行支持接口.SCM 的管理接口由并行组件框架调用,以便管理仿真组件.运行支撑接口供仿真引擎调用,处理仿真模型发送和接收的事件.每个仿真组件可以包含多个仿真实体.如图 2 所示,每个仿真组件对应的横线上的圆表示仿真实体.例如,仿真组件 component1 包括 4 个仿真实体,仿真组件 component2 有两个仿真实体.

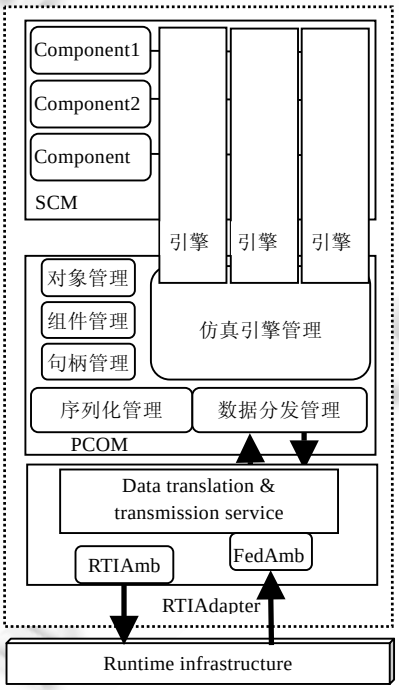


Fig.2 Parallel component framework

图 2 并行组件框架

© 中国科学院软件研究所 http://www.jos.org.cn

PCOM 是并行组件框架的核心模块.它通过组件管理服务、对象管理服务调用 SCM 的管理接口实现组件的加载、卸载、实例创建与删除、内存管理等功能.它的数据分发管理服务负责仿真实体之间的消息过滤和传递,提供高效的消息传输功能.同时,数据分发管理服务还负责与 RTI 适配器协调工作,将需要传递给其他成员的消息发送给 RTI 适配器,同时从 RTI 适配器中接收来自其他成员属性更新和交互的数据.仿真引擎(线程)的创建与删除、引擎之间的时间同步功能由 PCOM 的引擎管理服务提供.每个仿真引擎可以包含多个仿真实体,图 2 中表示有 3 个引擎,不同的引擎加载的实例是并行执行的.

RTIAdapter 包含 RTIAmb、FedAmb 和数据转换和传输服务(DTTS).RTIAmb 实现了联邦成员调用 RTI 服务的接口.RTIAdapter 通过 FedAmb 实现 RTI 的回调函数接口.DTTS 负责将管理框架发送的消息转换为属性更新或交互通过 RTIAmb 发送给 RTI,同时将收到的属性更新和交互转换为消息发给 PCOM 的数据分发管理服务,再由数据分发管理服务将消息发给仿真实体.

3.2 仿真组件模型设计

为了与联邦成员的开发方法保持一致,以便重用已有的联邦成员资源,仿真组件的设计采用了 HLA 的一些术语和概念,如更新属性值、发送交互等.根据联邦成员并行化和组件化的要求,为了支持仿真模型的动态组合和重用,仿真组件模型的设计必须满足如下的要求:仿真组件是被动式的,由仿真引擎通过调度离散事件的方式驱动运行;仿真组件间完全通过事件完成交互;为了达到动态组合的目的,可以不更改组件代码完成模型的组合;为了提高重用性和提供灵活的开发模式,组件可以是基于 RTI 的,也可以是非基于 RTI 的;组件间的数据需求采用类似 HLA 声明管理的方法描述.

SCM 的定义如下:

定义 1. SCM=⟨MPort,Attri,BaseModel,UserModel,ObjectBaseClass,InteractionBaseClass⟩.

其中,MPort,Attri,BaseModel,UserModel,ObjectBaseClass 和 InteractionBaseClass 分别表示 SCM 的管理接口、属性、模型基类、用户模型、对象基类和交互基类,如图 3 所示.组件管理接口为组件使用者提供了管理组件的功能.这些功能包括获取组件的属性,如 name,uid 等,创建、删除仿真实体.同时,仿真实体的查询、组件状态的保持和恢复也属于组件管理功能.组件管理接口是一组标准的函数,它在每个仿真组件都是相同的.组件属性 Attri 包含组件的名称,唯一标识和仿真实体集合.一方面,组件属性方便了对组件标识、查询和管理;另一方面,组件属性也为 PCOM 的数据分发管理服务对仿真实体之间进行消息过滤提供了额外的信息.

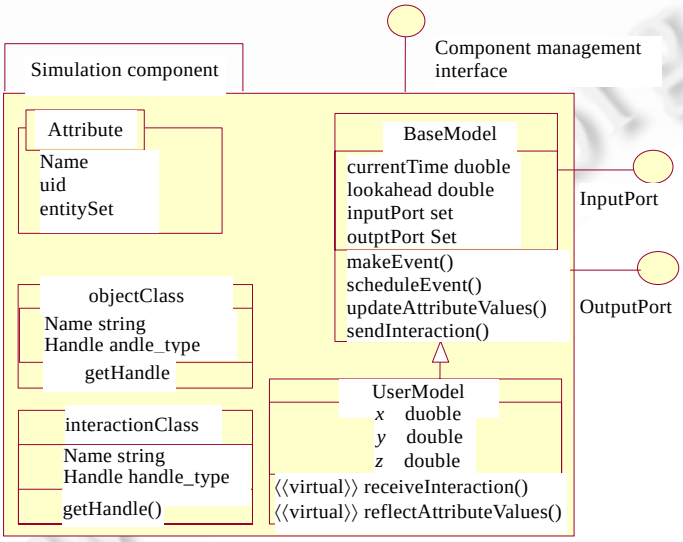


Fig.3 Simulation component model

图 3 仿真组件模型

© 中国科学院软件研究所 http://www.jos.org.cn

3.2.1 模型基类(BaseModel)

为了重用已有的联邦成员代码,仿真组件模型要尽可能少地或避免修改用户的代码,就能将用户代码移植成仿真组件.我们采用基于模型基类的方法来简化模型移植过程.每个仿真组件包含一个 BaseModel,它是所有UserModel 的根类.BaseModel 为 UserModel 提供仿真逻辑时间管理、事件规划与执行、数据发送与接受等功能.为了与联邦成员的开发方式尽可能地保持一致,BaseModel 提供了和 HLA 的对象管理服务类似的函数接口,如 *updateAtrributeValues*,*sendInteraction* 函数以及 *discoverObjectInstance*,*receiveInteraction*,*reflectAttributeValues* 等回调函数.BaseModel 的回调函数是纯虚函数,UserModel 必须实现这些函数.

无论是调用 *updateAtrributeValues* 更新仿真实体属性还是调用 *sendInteraction* 发送交互,BaseModel 都会将要发送的数据包装成事件,最终通过发送事件实现属性更新或发送交互的功能.为此,BaseModel 提供了非常完善的事件管理功能,例如创建事件、删除事件和规划事件等函数.

事件的定义如下:

定义 2. Event=⟨time,ID,port,value,executor,sender,eventType⟩.

其中,time,ID,executor,sender 和 eventType 分别表示事件发生的时间、标识、执行者、发送者和事件的类型.每个事件都有一个 port 属性和 value 属性.port 是 BaseModel 的数据输出端口.port 是一种命名化的通道,它的名字和它要传递对象类或交互类的名字必须相同.端口分为输入端口 inputPort 和输出端口 outputPort,如图 3 所示.输入端口表示仿真实体订购的数据类型.输出端口表示组件公布的数据类型.

端口定义如下:

定义 3. Port=⟨name,successor,predecessor,owner⟩.

其中,name 表示接口的名字,它与接口需要传递的数据类型的名字是一致的.name 通过带点号的类名表示类的层次关系;successor 表示接口的后继,即接收这个接口输出的数据的其他接口,它是一个接口集合;predecessor 表示接口的前导,即将数据传递给这个接口的接口集合;owner 表示接口的拥有者,也就是仿真实体.仿真实体会为它公布或订购的每个对象类和交互创建一个有相同名字的输出或输入接口.事件的 value 属性表示事件要传递的数据.value 可能是交互类或对象类.仿真组件通过输入/输出端口描述数据公布订购需求,这类似于 HLA 的声明管理功能.

我们以发送交互为例来说明 BaseModel 的工作原理.当一个仿真实体调用 *sendInteraction* 发送一个交互时,它首先会创建一个事件,将交互值赋予事件的 value 属性.然后,函数会根据交互类的类型,选择有相同名字的输出接口,并将输出接口的指针赋予事件的 port 属性.当事件完成初始化以后,它会被发送到 port 的 successor 属性的每一个 port.订购了该交互的仿真实体将会收到该事件,并将其保存起来,等待仿真引擎调度执行.仿真实体通过 BaseModel 执行事件,BaseModel 首先获取事件的 value 属性,然后将其作为参数,调用回调函数 *receiveInteraction*.

3.2.2 对象基类和交互基类

对象类和交互类与它们在 HLA 标准中的定义是一致的.无论是对象类还是交互类,每个类都有一个名字属性用于标识自己和句柄属性用于识别类之间的继承关系.根据 HLA 标准,如果一个成员订购了对象类的基类,它能收到该对象类的属性更新.类似地,仿真组件通过类名和仿真实体输入/输出端口的匹配达到相同目的.

3.2.3 组合组件

模型基类除了定义了输入/输出端口用于传递事件以外,还定义了管理子模型所需的属性和方法.如果一个仿真实体拥有子实体,它就是一个组合模型.通过模型基类,仿真实体可以拥有并管理子实体,可以动态增加、减少子实体.子实体之间的输入/输出端口耦合关系也是由其父仿真实体管理.通过动态改变实体的端口耦合关系,可以动态地改变仿真系统结构.

3.3 PCOM的管理服务

PCOM 的主要功能是要实现联邦成员内部仿真实体计算的并行化,为仿真实体提供线程级的并行计算能力.为了使不同引擎内的仿真实体能够并行计算,需要将实体之间的数据访问和函数调用封装成事件,由仿真引

© 中国科学院软件研究所 http://www.jos.org.cn

擎负责传递事件.为此,PCOM 需要提供对象管理服务、组件管理服务、数据分发管理服务、引擎管理服务以及其他的辅助服务.

图4描述了PCOM的各个服务之间的交互关系.步骤1表示PCOM的使用者请求创建一个仿真实体.PCOM通过对象管理服务处理这个请求.对象管理服务调用组件管理服务加载仿真组件.组件管理服务将仿真组件的句柄返回给对象管理服务,如图 4 中的步骤 2~步骤 5 所示.然后,对象管理服务调用仿真组件的管理接口创建仿真实体,仿真组件将该实例指针返回给对象管理服务,如图 4 中的步骤 6、步骤 7 所示.最后,对象管理服务通过调用数据分发管理服务处理仿真实体之间的数据过滤和传递关系来完成仿真实体的创建,如图 4 中的步骤 8、步骤 9 所示.引擎管理服务负责处理运行控制的请求,如开始、暂停、继续、停止、多次仿真等.收到开始仿真命令后,引擎管理服务从对象管理服务获得所有仿真实体的集合,并根据仿真实体的数量和机器的配置动态创建仿真引擎并给每个引擎分配实体.引擎管理服务开始仿真后还可以接收其他运行控制请求,直至仿真结束并返回运行状态,如图 4 的步骤 10~步骤 16 所示.

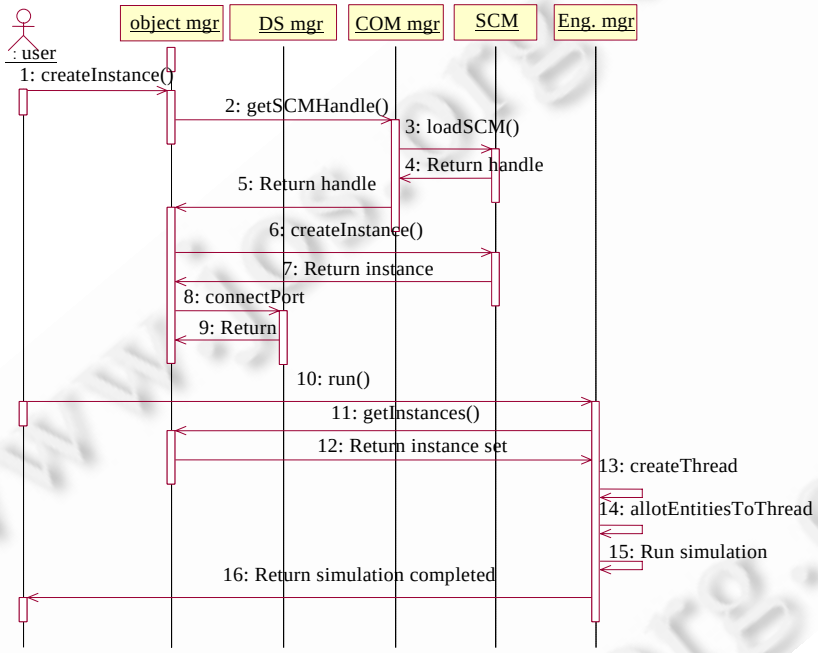


Fig.4 Interactions between services of PCOM

图 4 PCOM 中各个服务之间的交互过程

3.3.1 仿真引擎管理服务

多核平台的仿真引擎采用线程来实现,每个仿真引擎也称为逻辑进程.PCOM 的逻辑进程通过 3 个数据结构实现高效的事件排序和调度功能,这 3 个数据结构是 *event*,*scheduled_event_list* 和 *inputed_evnet_list*.*event* 是事件数据结构,如定义 2 所示.*scheduled_event_list* 是逻辑进程中的实体规划的事件列表,它采用了 Splay trees 数据结构,这是被公认最快的通用事件管理数据结构.*inputed_evnet_list* 是接收其他逻辑进程发送的事件列表.这个事件列表通过采用无锁队列数据结构解决多线程并发访问的问题.当逻辑进程可以进行时间推进以后,它从 *scheduled_event_list* 和 *inputed_evnet_list* 中取出安全事件进行重新排序,并按时戳顺序处理事件,保证仿真结果的正确性.除了规划执行事件以外,PCOM 的逻辑进程的还支持撤销已经调度的事件.

时间同步算法要确保逻辑进程的事件满足本地因果关系约束,即保证逻辑进程以时戳顺序处理事件.PCOM 目前实现了保守时间同步机制,最早的保守时间同步算法是 Chandy/Misra/Bryant 的空消息算法[20].为了解决空消息算法的缺陷,Chandy 等人提出了死锁检测和恢复算法.随后又出现栅障同步算法[21−24],栅障同步算

© 中国科学院软件研究所 http://www.jos.org.cn

法使用栅障同步协调逻辑进程之间的时间推进.当逻辑进程执行栅障同步时,它处于阻塞状态,并且在所有逻辑进程都执行完栅障同步之前一直处于阻塞状态.当所有处理器都执行了栅障同步时,栅障同步完成,并计算出每个逻辑进程可以安全推进的逻辑时间.栅障同步算法适合多处理器共享内存的计算平台.栅障同步算法有 3 种类别:集中式栅障同步、树栅障同步和蝴蝶栅障同步[12].目前,RTI 的保守时间同步算法主要实现了集中式栅障同步(如 KD-RTI[15],DMSO RTI[18])和树栅障同步(如 StarRTI,BH-RTI[17]).

PCOM 的仿真引擎管理服务实现了改进的集中式栅障同步算法.典型的栅障同步算法要求在所有的逻辑进程完成时间同步请求之前,先请求时间同步的逻辑进程处于阻塞状态.处于阻塞状态的逻辑进程不能处理事件.为了减少逻辑进程之间不必要的等待,我们改进了集中式栅障同步算法,如图 5 所示.逻辑进程请求时间同步后,并不进入阻塞状态,而是继续处理事件,用算法的 while 循环表示.在事件的处理过程中,可能还会产生新的事件,这些事件将保存在逻辑进程内部,而不是马上发给其他逻辑进程.此时,逻辑进程需要保存事件队列和模型的状态,因为它还有可能收到比已经处理的事件的时戳更小的事件.在处理时间过程中,一旦逻辑进程收到时间同步完成的信号,它将马上退出事件处理过程.然后检查事件队列中是否存在比已经处理的事件的时戳更小的事件,如果有,则需要进行回退操作,恢复事件队列和模型的状态,确保按照事件的时戳大小顺序处理事件.最后,逻辑进程根据同步算法返回的 LBTS 处理安全的事件.逻辑进程的 LBTS、GVT 的计算以及同步完成信号 synchronizationSignal 发送是由最后请求同步的逻辑进程负责完成.

```
Time synchronization algorithm.
Input: nextEventTime, LPID.
requestTime[LPID]:=nextEventTime
synchronizedLPCount++;
if (synchronizedLPCount=LPCount)
{
    calculate LBTS for each LP
    calculate GVT
    setSynchonizationSignal for each LP
}
else
{
    while (synchronizationSignal=false)
    {
            for (events in scheduled_event_list and inputed_evnet_list)
                {
                    if (timestamp of event≤nextEventTime)
                        process event;
                    endif
                }
    }
    If (minimize timestamp of event<timestamp of processed event)
        rollback processed events
     endif
    process events in scheduled_event_list and inputed_event_list whose timestamp less than LP's LBTS
}
endif
```

Fig.5 Time synchronization algorithm of logical process

图 5 逻辑进程时间同步算法

需要注意的是,在保守时间同步算法中,每个逻辑进程都有 *lookahead*.因为每个仿真实体都应该自己 *lookahead*,根据 *lookahead* 的定义,逻辑进程的 *lookahead* 应该是它运行的所有仿真实体 *lookahead* 的最小值.类似地,LBTS 的定义和计算方法参见文献[1].

3.3.2 组件管理服务

组件管理服务的主要功能是加载和卸载仿真组件(simulation component,简称 SC).一般情况下,组件管理服务辅助对象管理服务完成实例创建/删除、SC 查询等功能.当对象管理服务收到创建仿真实例的请求时,它会根据请求信息向组件管理服务查询 SC.如果组件管理服务当前没有加载这个 SC,那么它首先会加载这个 CS,然后

© 中国科学院软件研究所 http://www.jos.org.cn

将 CS 句柄回给对象管理服务.对象管理服务再通过 CS 的管理接口创建仿真对象实例.当仿真结束时,PCOM 需要通过组件管理服务卸载所有的 SC,释放 CS 资源.

3.3.3 对象管理服务

对象管理服务的主要功能包括对象实例的创建和删除、内存分配/回收等.对象管理服务通过组件管理服务获得已经加载的组件,并根据仿真组件实例化的要求动态地创建仿真实体.在创建仿真实例时,对象管理服务会根据组件的耦合关系,也就是组合组件的组合关系构造组合组件的仿真实体.此外,对象管理器可以根据要求动态创建对象实例,同时还可以动态地改变组合组件对象实例的组合关系.动态创建实例和变更组合关系是对组件管理器动态改变应用模型结构功能的进一步扩展.

对象管理服务通过调用 SCM 的管理接口实现创建对象实例功能.创建仿真实体时的内存分配发生在 SC 内部,对象管理服务只是获得仿真实体的指针.得到仿真实体后,对象管理服务会调用数据分发管理服务构造仿真实体与其他仿真实体之间的数据端口的连接关系,以确保仿真实体之间能够正确传递事件.

3.3.4 数据分发管理服务

在 HLA 标准中,联邦成员通过声明管理描述对象类和交互类的公布订购关系.声明管理服务使得联邦成员能在数据的发送端进行数据过滤,大大减少了网络上的数据流量,避免了类似 DIS 标准中消息数量随节点数呈指数增长的问题.此外,HLA 的数据分发管理服务通过定义的路由空间还能进一步减少消息的传播.数据分发管理是 HLA 的一个核心功能,也是一个研究难点.曲庆军等人研究了 HLA 数据分发管理的不同实现算法[25].史扬等人提出了基于路径空间层次划分的数据分发管理算法以及层次化组播地址分配策略[26,27].周忠等人研究了基于兴趣层次的相位过滤算法[28].从本质上说,HLA 的声明管理和数据分发管理都是为达到数据过滤,减少不必要的数据传输的目的.

仿真组件有 3 个层次的信息可以用于数据过滤,它们是仿真组件标识、实体标识和端口标识.基于这 3 个层次信息,数据分发管理服务实现了 3 种不同的数据过滤机制.

基于端口的数据过滤机制类似于 HLA 的声明管理.对象管理服务创建每个仿真实体时都会向数据分发管理服务注册仿真实体的端口,如图 4 中的步骤 8 所示.数据分发管理服务会根据端口的类型和名字建立端口之间的连接关系.数据分发管理服务将端口分为输入端口和输出端口两个集合.每个集合按照端口的名字组成一个树状结构,表示端口的继承关系.如图 6 所示,实线矩形表示输入端口,虚线矩形表示输出端口.当注册一个输入端口时,数据分发管理服务从输出端口集合找到和该输入端口名字相同的端口,然后将输入端口添加到每个输出端口的后继集合 successor 中,并将每个输出端口添加到输入端口的前导端口集合 predecessor 中.由于端口有继承关系,输入端口和输出端口的子端口也要建立连接关系,以便输入端口能够收到子输出端口发送的事件,如图 6 中的虚线箭头所示.如图 6 所示,输入端口 object.entity 不仅可以收到输出端口 object.entity 发送的事件,因为端口的继承关系,输入端口 object.entity 还可以收到输出端口 object.entity.agent 和输出端口 object.entity.weapon 发出的事件.通过端口过滤机制,仿真实体不用在接收到事件后再判断是否自己需要的事件,而是在事件的输出端实现数据过滤.

基于实体标识的数据过滤是一种运行时数据过滤机制.在端口数据过滤机制下,用户模型调用 *sendInteration* 发送交互,所有具有与该交互所属交互类同名的输入端口都会收到一个 RECEIVE_INTER_MSG 事件.但是,仿真组件的仿真模型基类 BaseModel 提供的 *sendInteration* 函数还支持向特定的仿真实体发送交互.为了向指定的仿真实体发送交互,只需在调用 *sendInteraction* 函数时提供该实体的标识.如果仿真实体在发送事件时提供了目标实体标识,那么数据分发管理服务只向该实体发送事件.其他仿真实体即使订购了该类型的事件也不会收到该事件.属性更新函数 *UpdateAttributeValues* 也提供类似的功能.在仿真实体数量众多时,基于实体标识的运行时数据过滤机制能够非常有效地减少事件数量.

基于仿真组件标识的数据过滤机制是一种运行前通过配置实现的数据过滤机制.只要输入端口和输出端口的名字匹配,基于端口的数据过滤机制就会建立它们之间的连接关系,而不考虑它们属于哪个仿真组件.在连接输入/输出端口过程中,数据分发管理服务不但可以获得端口的标识,而且可以获取端口所属仿真实体的标识

© 中国科学院软件研究所 http://www.jos.org.cn

以及仿真实体所属仿真组件的标识.数据分发管理服务可以利用仿真组件标识通过一个配置文件指定仿真组件的需求关系,也就是说,指定一个仿真组件的实体不接收哪些仿真组件的实体发送的事件.基于仿真组件标识的数据过滤机制的优先级比基于端口的数据过滤机制的优先级要高.即使输入/输出端口的名字匹配,但是因为输入/输出端口分别属于两个不需要建立端口连接仿真组件的仿真实体,所以不能建立端口之间的连接关系.

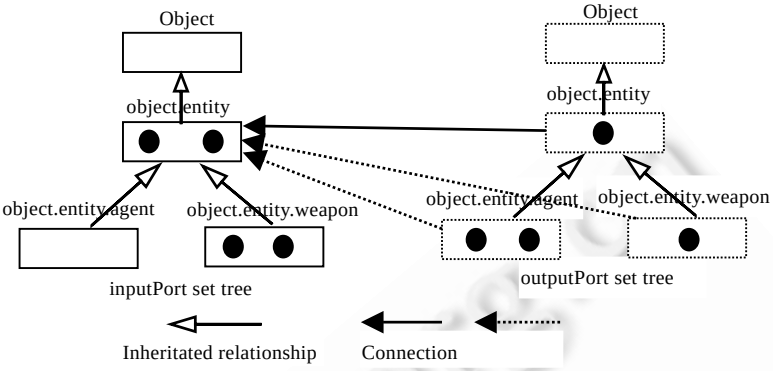


Fig.6 Port-Based data distribution mechanism

图 6 基于端口的数据分发机制

通过这 3 种数据过滤机制,数据分发管理服务能够为大规模的仿真实体提供高效的事件传递功能.这 3 种数据分发管理机制可以同时使用,也可以单独使用.基于端口的数据过滤机制和基于实体标识的数据过滤机制与具体的仿真系统无关,是通用的数据过滤机制.基于仿真组件标识的数据过滤机制一般在针对具体的系统性能优化时使用.

3.4 RTI适配器

虽然仅使用 PCOM 就能支持仿真组件在多核平台上进行并行仿真,但是为了能够与其他的联邦成员组成联邦,并行组件框架还需要与 RTI 进行交互.这个功能由 RTIAdapter 实现.RTIAdapter 通过 RTIAmb 调用 RTI 服务,并通过 FedAmb 实现 RTI 的回调函数和 RTI 进行交互.此外,RTIAdaper 通过数据转换和传输服务(DTTS)模块负责和 PCOM 进行交互.RTIAdapter 的结构如图 2 所示.RTIAdapter 要实现普通联邦成员的功能,包括公布订购声明、注册对象实例、更新属性、发送交互、反射属性、接收交互和时间同步.

当数据分发管理服务收到一个注册端口的请求后,除了在 PCOM 内部连接输入/输出端口以外,还需要以端口名字作为参数向 RTIAdapter 发出公布订购声明请求.RTIAdpater 收到请求后,调用 RTI 的声明管理函数完成对象类、交换类的公布订购声明.

HLA 标准规定 RTI 不提供数据打包解包服务,只负责数据的传递.数据的打包解包由联邦成员进行协定.在仿真运行时,PCOM 的数据分发管理服务会将仿真实体的 UPDATE_ATT_MSG 和 SEND_INTER_MSG 类型事件发送给 RTIAdpater 的 DTTS.DTTS 收到事件后提取事件的 value 属性,即对象实例的属性值或交互的参数值.如果是一个 UPDATE_ATT_MSG 事件,那么 DTTS 会将 value 赋予一个 *AttributeHandleValueMap* 变量,然后调用 RTI 的 *updateAttributeValues* 更新属性值.如果是 SEND_INTER_MSG 事件,DTTS 则会将 value 赋予一个 *ParameterHandleValueMap* 变量,然后调用 RTI 的 *sendInteraction* 函数发送交互.如果是首次更新对象实例的属性值,DTTS 首先向 RTI 注册实例,然后再更新属性值.当 RTIAdapter 收到 RTI 的 *reflectAttributeValues* 回调以后,DTTS 构造一个 REF_ATT_MSG 事件,并将 *AttributeHandleValueMap* 变量表示的属性值转换为事件 value 属性,然后发给 PCOM 的数据分发管理服务.RTIAdapter 收到 RTI 的 *receiveInteraction* 回调后,处理过程与收到 *reflectAttributeValues* 回调的处理过程类似.

除了数据转换和传输功能以外,RTIAdapter 还需要协调 RTI 和 PCOM 的时间管理.当 PCOM 的仿真引擎管

© 中国科学院软件研究所 http://www.jos.org.cn

理服务收到所有逻辑进程的时间推进请求后,它能够得到所有逻辑进程的请求时间的最小值.这个最小值就是并行联邦成员向 RTI 请求推进的逻辑时间.RTIAdapter 用这个时间向 RTI 请求时间推进.并行联邦成员的 lookahead 是并行组件框架中所有仿真实体 lookahead 的最小值.

4 性能监测和动态负载平衡

在第 3 节介绍 PCOM 时,我们提到仿真引擎管理服务对象管理服务获得仿真实体,并将仿真实体分配给逻辑进程,如图 4 中的步骤 10~步骤 14 所示.这是关于负载分配的问题.负载分配有静态和动态分配两种方式.由于模型负载的不平衡和仿真运行的动态性,静态负载分配不能满足高性能仿真的要求,需要动态负载平衡方法.在并行分布仿真领域,动态负载平衡并不是一个新概念.较早研究仿真动态负载平衡的是 Reiher 等人[29].他们提出了基于处理器效用(effective utilization of processor)的 Time Warp 仿真系统动态负载平衡方法.Glazer 等人采用控制分配给逻辑进程 CPU 时间片以减少逻辑进程之间逻辑时间差的方法来获得逻辑进程的负载平衡[30].在此基础上,Jiang 等人通过预测未来可用 CPU 时间改进逻辑进程的 CPU 时间分配,协调逻辑进程的逻辑时间,减少回退的次数[31].Cai 等人研究了基于网格的 HLA 负载平衡管理系统[32].Azzedine 等人在基于 HLA 的大规模仿真系统动态负载平衡方面做了大量的工作[33−37].他们提出一种可扩展的层次动态负载平衡构架.这种构架在本地和集群两层检测 HLA 仿真系统运行负载变化,以发现联邦成员之间的计算负载不平衡和通信负载不平衡,并运用联邦成员迁移的方法减少成员之间的通信响应时间和平衡节点之间的计算负载.

在原理上与前面的提出的动态负载平衡方法类似,PCOM 的动态负载平衡也同样要解决检测不平衡状态和负载迁移的问题,但是具体的方法也有不同之处.因为 PCOM 的所有逻辑进程都是运行在同一个物理进程空间内部,所以只能通过检测不同逻辑进程占用的 CPU 时间的方式来获取逻辑进程的负载数据.为了更加精确地进行负载平衡,除了获取逻辑进程占用 CPU 时间以外,我们还需要检测每个仿真实体处理仿真事件所需的 CPU 时间.因为同一个物理进程内部的线程通信的开销非常小,甚至可以忽略不计,我们不需要检测逻辑进程的通信负载.一般说来,动态负载平衡调度可分为集中式调度和分布式调度[33].在 PCOM 中,所有的仿真实体、逻辑进程都处于同一进程空间,适合采用集中式负载平衡调度.

4.1 性能监测

动态负载平衡的目的是减少仿真系统的运行时间和通信响应时间,提高系统运行的性能.运行时间的变化是评估动态负载平衡方法是否有效的一个最重要的指标.对于同一个的仿真任务,假设没有采用负载平衡时的运行时间为 H,采用动态负载平衡后的运行时间为 H_b,那么系统性能的变化定义为

$$\beta=(H-H_b)/H \tag{1}$$

如果β>0,则称为性能提升;反之则称为性能下降.为了便于讨论,假设并行联邦成员有 n 个逻辑进程并保持不变,一共执行 m 次循环完成仿真.对于基于保守时间同步的并行离散事件仿真,逻辑进程不断地循环执行处理事件和请求时间推进两个步骤,直至仿真结束.假设逻辑进程完成一个循环需要的物理时间为 h_{ij},i 为逻辑进程编号,j 为仿真的循环编号.由于采用保守时间同步方法,系统完成一次仿真循环的物理时间由最慢的逻辑进程决定,所以有

$$H=\sum_{j=1}^{m}\mathrm{Max}_{i=1,2,\ldots,n}(h_{ij}) \tag{2}$$

如果采用动态负载平衡,假设完成仿真一共进行了 l 次迁移,监测性能数据以及每次迁移消耗的时间为 M_k,

$$H_b=\sum_{j=1}^{m}\mathrm{Max}_{i=1,2,\ldots,n}(h_{ij})+\sum_{k}^{l}M_k \tag{3}$$

其中,h_{ij} 是处理事件所需时间 p_{ij} 和请求时间推进所需时间 a_{ij} 之和.a_{ij} 为逻辑进程 i 开始请求时间推进到被允许推进的时间间隔.除了要记录 p_{ij},a_{ij} 和 M_k 以外,还要获取仿真实体 s 在每次仿真循环中处理事件需要消耗的 CPU 时间,并计算其平均 CPU 时间消耗 t_{sj}.仿真实体的平均 CPU 时间消耗 t_{sj} 是仿真实体 s 每次消耗 CPU 时间之和除以仿真循环的次数.

© 中国科学院软件研究所 http://www.jos.org.cn

4.2 基于实体迁移的动态负载平衡

增加动态负载平衡功能要求逻辑进程在每个仿真循环记录处理事件的时间 p_{ij} 和请求时间推进的时间 a_{ij}.每个仿真实体在处理事件时要记录处理事件的时间.在仿真循环进入时间同步阶段,由仿真引擎管理服务为仿真实体计算平均 CPU 时间消耗 t_{sj}.仿真实体被保存在一个按 t_{sj} 增序排列的实体列表 *tep_list* 中.排在 *tep_list* 前面的仿真实体称为低耗时实体,排在队列后面的仿真实体称为高耗时实体.每个逻辑进程都有一个 *tep_list* 记录它的负载的仿真实体.类似地,仿真引擎管理服务也将逻辑进程记录在在一个按 p_{ij} 增序排列的逻辑进程列表 *lp_tep_list* 中.排在前面的称为低负载逻辑进程,排在后面的称为高负载逻辑进程.*lp_tep_list* 记录的数据在每个仿真循环都会更新.为了使逻辑进程的负载尽可能地平衡,我们提出了基于仿真实体迁移的 min-max- exchange balancing algorithm,如图 7 所示.

```
Min-Max-Exchange balancing algorithm.
Input: lp_tep_list.
do
    low_lp   getNextUnderloadedLP(lp_tep_list)
    high_lp   getNextOverloadedLP(lp_tep_list)
    load_dif   high_lp–low_lp
    if (load_dif >Min)
        do
            low_entity getNextLowTimeConsumer(low_lp)
            high_entity getNextHighTimeConsumer(high_lp)
            load_dif =load_dif–(high_enity–low_entity)
            if (load_dif >0)
               exchangeEntityforLP(low_entity,high_entity)
            end if
        while (load_dif >0)
    end if
while (high_lp–low_lp>Min)
```

Fig.7 Min-Max-Exchange balancing algorithm

图 7 Min-Max-Exchange 负载平衡算法

算法从队列 *lp_tep_list* 两端分别获取最低负载逻辑进程和最高负载逻辑进程并计算它们的负载差值.如果负载差值大于一个设定的阈值 Min,则交换它们的仿真实体,进行负载平衡.首先,让低负载逻辑进程的最低耗时实体 *low_entity* 和高负载逻辑进程的最高耗时实体 *high_entity* 进行比较.如果它们的耗时差值小于两个逻辑进程的负载差值,则进行实体迁移.实体迁移后更新逻辑进程的负载差值.如果负载差值大于 0,则继续交换剩下的仿真实体,直到负载差值不大于 0.然后,算法对下一对逻辑进程进行负载平衡,遍历完所有逻辑进程后结束.因为算法首先对最小负载和最多负载逻辑进程进行负载平衡,并且首先交换耗时最少和最多的实体,因此称为 min-max-exchange balancing algorithm.负载平衡需要迁移仿真实体,这个时候模型不能处理事件.这就是负载平衡开销.因此,负载平衡算法必须足够简单和有效,时间开销要尽可能地小.如果算法太耗时,则反而会抵消负载平衡的效果.阈值 Min 的设置也很关键,如果 Min 太小则会导致频繁的负载迁移;如果太大,又不能有效地触发负载平衡.

逻辑进程之间的动态负载平衡通过仿真实体的迁移实现.负载平衡和组件管理框架向 RTI 请求时间同步进行.组件管理框架调用 RTI 的 *TimeAdvanceRequest* 时,所有的逻辑进程处于阻塞状态.这时,可以安全地更变逻辑进程的模型列表和事件列表.转移逻辑进程的仿真实体的同时,要将与该仿真实体关联的事件转移到目标逻辑进程.源逻辑进程和目标进程的下一个事件的时刻点也要重新计算.

5 实验和结果分析

为了评估并行组件框架在提高仿真系统性能方面的效果,我们设计和实现了一系列的实验.实验平台是一组联网的计算机,有 8 个节点.节点处理器为 Quad-core 3.00 GHz Intel Core CPU,内存为 4GB RAM.节点之间通过千兆以太网连接.操作系统是 Windows XP,编译环境是 visual studio 2008.RTI 软件采用 KD–RTI 1.3NG.RTI 服

© 中国科学院软件研究所 http://www.jos.org.cn

务器部署在其中的一个节点上.

测试数据的仿真实体数量在 640~10 240 个之间变化.每个仿真实体都有一定的计算负载.最小的负载和最大的负载相差 50 倍.仿真实体的负载大小是随机的,在最大和最小负载之间服从均匀分布.

实验中,每个联邦成员都是既时间控制又时间受限.联邦成员采用时间步进的请求时间推进方式,普通成员的步长都相同,为 1;并行成员没有固定的步长,按照事件的方式请求时间推进,时间前瞻量都是 1,每次实验的仿真时间为 0~5 000s.每个联邦成员都相互公布订购仿真实体的属性,也就是联邦成员之间相互发送和接收仿真实体的属性值.

5.1 普通联邦成员和并行联邦成员的测试结果比较

为了比较并行联邦成员和普通联邦成员的性能,对相同数量的仿真实体分别用并行成员和普通成员进行仿真,然后比较系统的运行时间.对于普通成员,测试增加联邦成员数量能否提高系统的性能;对于并行联邦成员,评估改变逻辑进程数量对系统性能的影响.为了便于比较,对于相同数量的仿真实体,分别用 8,16,32,64 个普通成员构成的联邦进行测试.在每次测试中,仿真实体被平均分配到联邦成员中.每个计算节点的联邦成员个数也是相同的.在一个节点运行多个联邦成员是将仿真实体分配到不同的联邦成员,减少每个联邦成员的仿真实体数量.通过 RTI 同步,多个联邦成员可以充分利用多核处理器的计算能力减少联邦成员的计算开销.但是,这会增加联邦成员之间的通信开销,能否提升联邦的运行性能还要看实验结果.对于并行联邦成员,每个节点只部署一个联邦成员.也就是说,无论有多少仿真实体,都只有 8 个并行联邦成员运行.但是,我们通过改变并行联邦成员的逻辑进程数量来测试并行度的变化对系统性能的影响.为了便于比较,分别测试了并行联邦成员的逻辑进程为 1,2,4,8 的情况.并行联邦成员的仿真实体也是在初始化时平均分配到每个逻辑进程,没有动态负载平衡.

实验的结果如图 8 所示.其中,x 轴表示仿真实体的数量,y 轴是仿真运行时间.图 8 的曲线显示,无论是普通成员还是并行成员,仿真运行时间都随仿真规模的增大而增大.从普通成员的测试结果曲线可以看出,无论仿真实体数量是多少,在仿真实体数量相同时,增加联邦成员以减少每个成员运行的仿真实体数都不能有效减少整个仿真系统的运行时间,如图 8 中的 8,16,32,64 federates 曲线所示.这是因为虽然增加联邦成员可以减少模型运行时间,但是联邦成员之间的通信和请求时间推进开销同时也增大了.

从图 8 的 1,2,4,8 Logical Processes 曲线可知,对于相同数量的仿真实体,仿真运行时间随并行联邦成员的逻辑进程增多而减少.增加逻辑进程可以并行处理更多的仿真事件,仿真模型运算的并行度增加.一个逻辑进程的并行联邦成员和传统联邦成员的性能相差不大,证明并行组件框架带来的额外开销很小.当仿真规模较大时,两个逻辑进程的并行成员大约能够提升性能 25%,4 个逻辑进程能够提升性能 35%,而 8 个逻辑进程能够提升性能 45%.

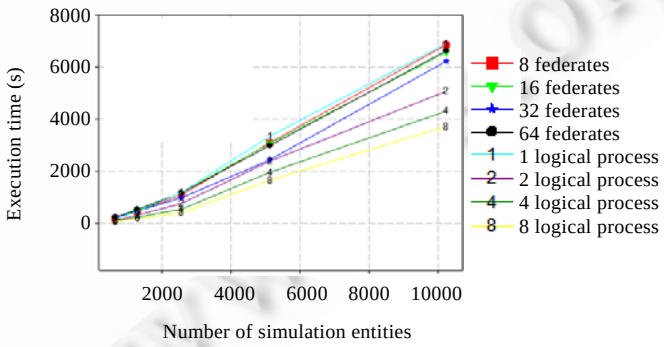


Fig.8 Execution time for different number of simulation entities

图 8 不同数量仿真实体的系统运行时间

图 9 表示不同数量的联邦成员分别运行 640 个仿真实体的时间开销.其中,x 轴表示运行时间,y 轴表示联邦

© 中国科学院软件研究所 http://www.jos.org.cn

成员的数量.仿真运行时间由仿真实体运行时间及同步和通信时间两部分构成.图 9 清晰地描述了:随着联邦成员数量的增加,实体运行时间减少,但是联邦成员之间的通信和请求时间推进开销却明显增加.最终的仿真时间变化不大.这解释了图 8 中显示的结果:增加联邦成员数量的方式,减少每个联邦成员的仿真实体数量并不能减少系统的运行时间.

图 10 描述了每个并行联邦成员运行 1 280 个仿真实体时的运行时间与逻辑进程数量之间的关系.整个联邦中有 8 个并行成员,运行时间是 8 个成员的平均值.从图 10 可以看出:随着逻辑进程数量的增加,系统的运行时间逐渐减少;但是,联邦成员之间的同步和通信开销不随逻辑进程数量的变化而改变;仿真实体的运行时间随着逻辑进程的增加而显著减少,从而使系统运行时间减少.这也解释了在图 8 中当仿真实体数量较少时,并行成员并不能有效提高系统性能的原因.

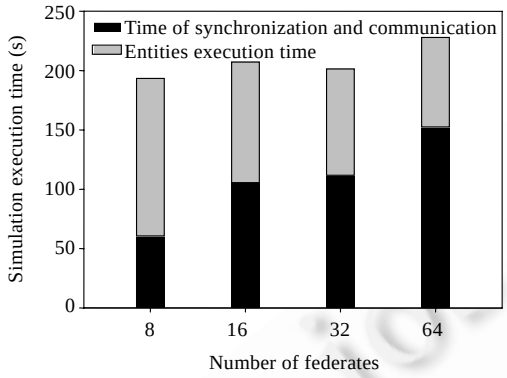


Fig.9 Execution time analysis of normal federate

图 9 普通成员运行时间分析

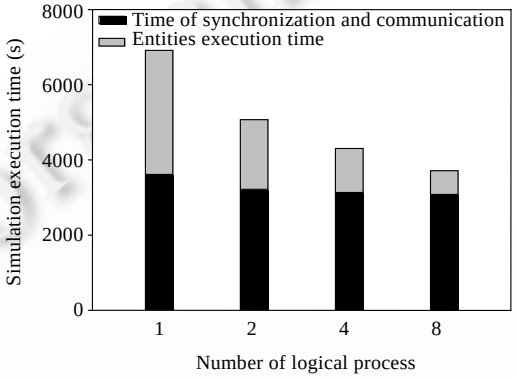


Fig.10 Execution time analysis of parallel federate

图 10 并行成员运行时间分析

5.2 并行成员和并行成员加负载平衡实验结果比较

在这个实验中,我们通过比较有负载平衡功能的并行成员和没有负载平衡功能的联邦成员运行时间,研究负载平衡的效果.在仿真初始化时,仿真实体被平均分配给逻辑进程.仿真开始后,负载平衡算法监测逻辑进程的负载差异.如果逻辑进程的负载差异大于阈值 Min,则在逻辑进程间迁移实体进行负载平衡.阈值 Min 的设置与具体的应用相关,在这个实验中 Min 取 8.我们分别研究了每个并行成员 4 个逻辑进程和 8 个逻辑进程时负载平衡算法的效果.联邦成员运行 80~1 280 个仿真实体.图 11 是 4 个逻辑进程的实验结果.其中,左边的 y 轴表示仿真运行时间,右边的 y 轴是发生迁移的次数,x 轴是仿真实体的数量.

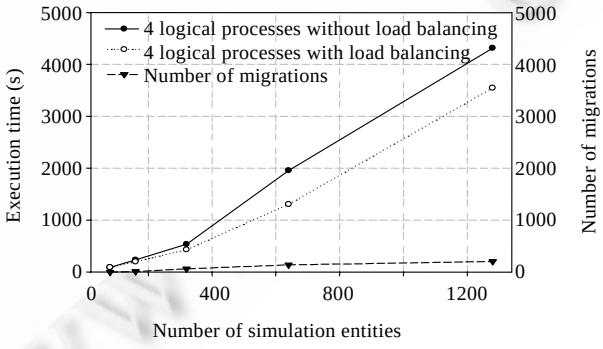


Fig.11 Load balancing of parallel federate with 4LPs

图 11 4 个逻辑进程的并行成员负载平衡

© 中国科学院软件研究所 http://www.jos.org.cn

从图 11 可以看出,当仿真实体数量较少时,逻辑进程之间的负载差异没有超过阈值 Min,没有启动负载平衡.此时,负载平衡算法的开销不大,有负载平衡功能的成员和没有负载平衡功能的成员的运行时间相差不大.随着仿真实体的增多,逻辑进程之间的负载差异变大,进行负载平衡的次数增多.有负载平衡功能的联邦成员的运行时间要比没有负载平衡功能的联邦成员运行时间明显减少.发生负载平衡的次数为 0~200 次左右.图 12 是每个联邦成员 8 个逻辑进程的实验结果,效果与每个联邦成员 4 个逻辑进程的情况类似,出现负载平衡的次数稍多,为 0~300 次左右.从图 11、图 12 中的实验结果可知,有负载平衡功能的并行成员的性能比没有负载平衡功能的并行成员大约提升了 17%.

结合前面的测试结果,当仿真实体较多时,与普通联邦成员相比,包含 8 个逻辑进程并且有负载平衡功能的并行联邦成员大约能够使仿真系统的性能提升 56%.

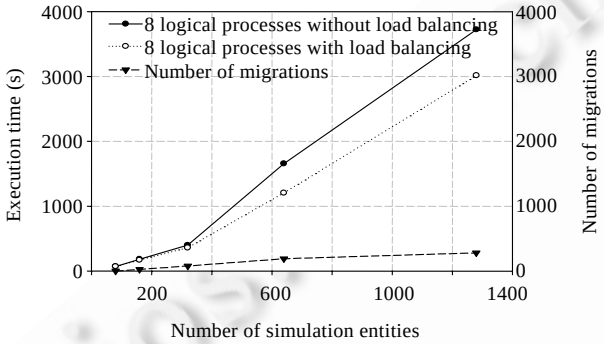


Fig.12 Load balancing of parallel federate with 8LPs

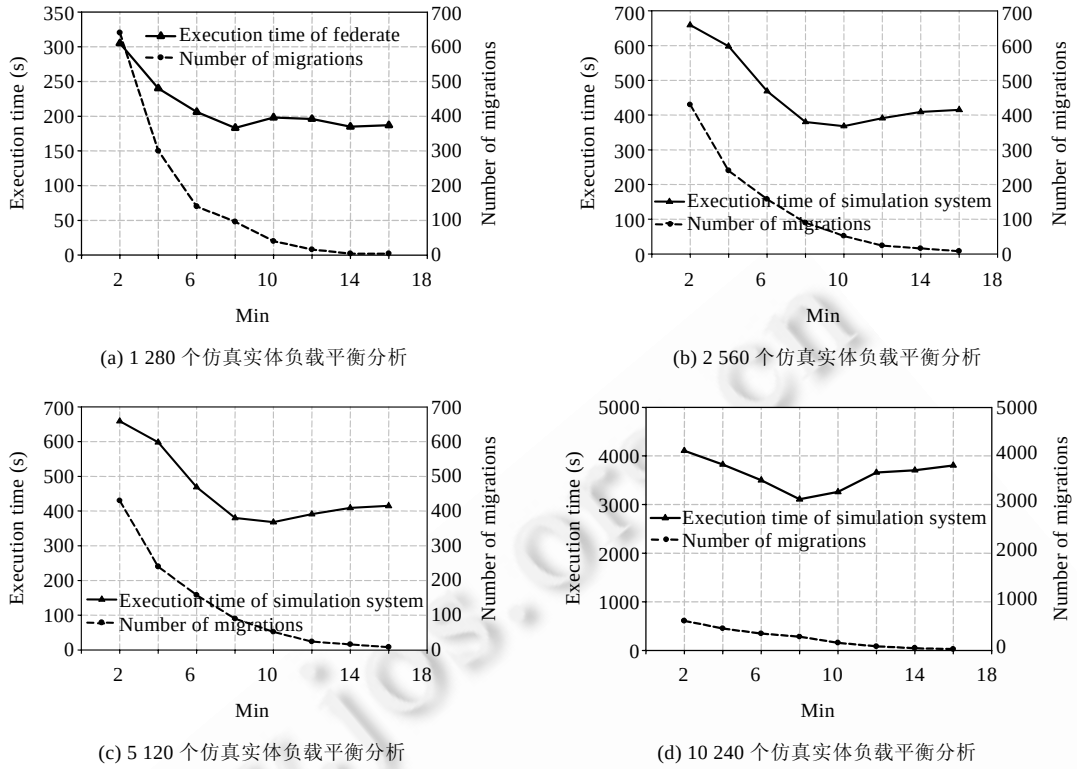
图 12 8 个逻辑进程的并行成员负载平衡

5.3 负载平衡阈值Min分析

阈值 Min 的设定对负载平衡算法效果有很大的影响,我们需要根据仿真系统运行时间的变化来确定 Min.为了分析 Min 与仿真运行时间和迁移次数的关系,我们分别对不同数量仿真实体的仿真系统进行实验.在实验中,我们通过改变 Min 的大小并记录相应的仿真运行时间的方法来确定最优的 Min.同时,通过对比具有不同数量仿真实体实验的最优 Min,研究设置 Min 的方法.实验结果如图 13 所示.实验中的每个并行成员都是 8 个逻辑进程.图 13(a)~图 13(d)分别是 1 280,2 560,5 120,10 240 个仿真实体的仿真实验结果.其中,x 轴是阈值 Min,左边的 y 轴是仿真运行时间,右边的 y 轴是发生负载迁移的次数.所有实验的 Min 值取 2,4,6 等几个典型值.从图 13(a)中可以看出 Min 值与仿真运行时间的关系.当 Min 从 2 到 8 变化时,仿真运行时间逐渐减少.当 Min 大于 8 以后,仿真运行时间保持稳定,不再减少.负载迁移的次数随着 Min 的增大而逐渐减少.当 Min 大于 16 时,负载迁移次数接近于 0.这说明当 Min 大于 16 时,负载平衡算法已经不能检测出逻辑进程之间的负载不平衡.图 13(b)~图 13(d)与图 13(a)有相似的结论,当 Min 的取值接近 8 时,仿真运行时间达到最小值.这也是我们在第 5.2 节中将 Min 设置为 8 的原因.

从图 13 还可以看出,Min 的取值很重要.对于实验模型,当 Min 小于 8 时,会导致负载迁移波动,累积的负载平衡开销使得最终的仿真运行时间有可能比没有负载平衡功能的运行时间要长.当 Min 太大时,又无法有效地进行负载平衡.Min 的取值与具体仿真系统的模型相关,每个系统应该至少有 1 个最优的 Min 值,这个值可以通过实验方法获得.此外,通过比较图 13(a)和图 13(b)~图 13(d)图可知,通过选取少量有代表性的模型实验,可以得到整个仿真系统近似最近 Min 值.这对于大规模复杂系统来说很有意义,因为对于模型数量众多的系统,获取最优 Min 值很耗时.

© 中国科学院软件研究所 http://www.jos.org.cn



(a) 1 280 个仿真实体负载平衡分析 (b) 2 560 个仿真实体负载平衡分析

(c) 5 120 个仿真实体负载平衡分析 (d) 10 240 个仿真实体负载平衡分析

Fig.13 Relation between Min and execution time, number of migrations

图 13 Min 与运行时间、迁移次数的关系

6 结论和下一步的工作

在本文中,我们提出了面向仿真组件的并行联邦成员框架,以充分利用多核处理器的计算能力提高联邦成员的运行速度.并行成员框架允许仿真开发者以组件的方式开发仿真模型.通过并行联邦成员框架管理服务的支持,仿真模型可以在成员内并行执行,缩短了成员的运行时间.仿真组件实现了 RTI 的属性更新、发送交互函数以及回调函数,保证了仿真模型开发与传统联邦成员开发相似,降低了模型开发的难度.并行联邦成员框架为仿真模型提供时间管理服务,并与 RTI 协同时间推进,使仿真模型能够并行执行.实验结果表明,并行联邦成员能够有效地减少仿真模型的运行时间,提高仿真系统的性能.此外,并行联邦成员框架还有负载平衡功能,进一步优化并行联邦成员的运行时间.实验结果表明,并行联邦成员框架的动态负载平衡功能能够监测逻辑进程之间存在的负载不平衡,并通过仿真实体迁移实现逻辑进程的动态负载平衡,进一步减少成员运行时间.总之,采用基于组件的并行联邦成员框架可以为基于 HLA 的仿真系统带来如下的好处:(1) 基于组件的联邦成员开发,支持组件组合;组件提供与 RTI 的对象管理相类似的接口,提高模型的可重用性.(2) 在联邦成员内实现仿真模型的并行执行,减少仿真系统的运行时间.(3) 具备联邦成员内动态负载平衡能力,消除由于负载不平衡而导致的逻辑进程之间相互等待的时间开销.

下一步的工作是深入分析负载平衡算法的时间开销,进一步优化系统的性能,并研究自动调节阈值 Min 的方法.

© 中国科学院软件研究所 http://www.jos.org.cn

References:

[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. New York: Institute of Electrical and Electronics Engineers Inc., 2000. 1−3.

[2] Borkar SY, Dubey P, Kahn KC, Kuck DJ, Mulder H, Pawlowski SS, Rattner JR. Platform 2015: Intel Processor and Platform Evolution for the Next Decade. Santa Clara: Intel Corporation, 2005. 1−12.

[3] Gong JX, Han C, Qiu XG, Huang KD. Constructing the extensible HLA federate architecture. Journal of System Simulation, 2006,18(11): 3126−3130 (in Chinese with English abstract).

[4] Radeski A, Parr S, Keith-Magee R. Component-Based development extensions to HLA. In: Severinghaus R, ed. Proc. of the 2002 Spring Simulation Interoperability Workshop. Washington: IEEE Press, 2002. 190−195.

[5] Alex R, Shawn P. Towards a simulation component model for HLA. In: Severinghaus R, ed. Proc. of the 2002 Fall Simulation Interoperability Workshop. Washington: IEEE Press, 2002. 216−221.

[6] Katarzyna R, Marian B, Peter MA. Sloot dynamic interactions in HLA component model for multiscale simulations. In: Bubak M, ed. Proc. of 8th Int'l Conf. on Computational Science (LNCS ICCS 2008). Berlin, Heidelberg: Springer-Verlag, 2008. 217−226. [doi: 10.1007/978-3-540-69387-1-24]

[7] Benali H, Bellamine N, Saoud B. Towards a component-based framework for interoperability and composability in modeling and simulation. Trans. of the Society for Modeling and Simulation Int'l, 2011,87(1-2):133−148. [doi: 10.1177/ 0037549710373910]

[8] Farshad M, Peder N, Rassul A. Simulation model composition using BOMs. In: Alba E, ed. Proc. of the 10th IEEE Int'l Symp. on Distributed Simulation and Real-Time Application. Washington: IEEE Computer Society Press, 2006. 242−252. [doi: 10.1109/ DS-RT.2006.34]

[9] Farshad M, Rassul A, Gary T. A rule-based approach to syntactic and semantic composition of BOMs. In: Roberts DJ, ed. Proc. of the 11th IEEE Int'l Symp. on Distributed Simulation and Real-Time Application. Washington: IEEE Computer Society Press, 2007. 145−155. [doi:10.1109/DS-RT.2007.10]

[10] Pettry MD, Weisel EW, Mielke RR. Composability theory overview and update. In: Sharp R, ed. Proc. of the 2005 Spring Simulation Interoperability Workshop. Washington: IEEE Press, 2005. 12−17.

[11] Jefferson DR. Virtual time. ACM Trans. on Programming Languages and Systems, 1985,7(3):404−425. [doi: 10.1145/3916.3988]

[12] Fujimoto R. Parallel and Distributed Simulation Systems. New York: John Wiley & Sons, Inc., 2000. 65−71.

[13] Steinman JS. Interactive SPEEDES. In: Zobrist GW, ed. Proc. of the 24th Annual Symp. on Simulation. Washington: IEEE Computer Society Press, 1991. 149−158. [doi: 10.1145/106073.306852]

[14] Steinman JS, Douglas RH. Evolution of the standard simulation architecture. In: Alberts DS, ed. Proc. of the Command and Control Research Technology Symp. Washington: IEEE Press, 2004. 268−274.

[15] Huang J, Hao JG, Huang KD. The research and application of HLA-based distributed simulation environment KD-HLA. Journal of System Simulation, 2004,16(2):214−220 (in Chinese with English abstract).

[16] Lib BQ, Wang HM, Yao YP. A non-deadlock time management algorithm. Journal of Software, 2003,14(9):1515−1522 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1515.htm

[17] Lü F, Zhou Z, Wei S, Wu W. Time management of RTI based on negotiation group. Journal of Beijing University of Aeronautics and Astronautics, 2008,34(9):1084−1087 (in Chinese with English abstract).

[18] Department of Defense, Defense Modeling and Simulation Office. RTI 1.3—Next Generation Programmer's Guide. Version 3.2. Alexandria: Defense Modeling and Simulation Office, 2000. 3.1−3.10.

[19] Su NL, Li Q, Wang WP. Parallelized modification of interaction modes among component-based simulation model. Systems Engineering and Electronics, 2010,32(9):2015−2020 (in Chinese with English abstract).

[20] Chandy KM, Misra J. Distributed simulation: A case study in design and verification of distributed programs. IEEE Trans. on Software Engineering, 1978,5(5):440−452. [doi: 10.1109/TSE.1979.230182]

[21] Konas P, Yew PC. Parallel discrete event simulation on shared-memory multiprocessors. In: Rutan AH, ed. Proc. of the 24th Annual Simulation Symp. Washington: IEEE Computer Society Press, 1991. 134−148. [doi: 10.1145/106073.306851]

© 中国科学院软件研究所 http://www.jos.org.cn

[22] Chandy KM, Sherman R. The conditional event approach to distributed simulation. In: Fujimoto RM, ed. Proc. of the SCS Multiconference on Distributed Simulation. Tampa: SCS Press, 1989. 93−99.

[23] Loucks WM, Preiss BR. The role of knowledge in distributed simulation. In: Nicol D, ed. Proc. of the SCS Multiconference on Distributed Simulation. San Diego: SCS Press, 1990. 9−16.

[24] Nicol DM. The cost of conservative synchronization in parallel discrete event simulations. Journal of the ACM, 1993,40(2): 204−333. [doi: 10.1145/151261.151266]

[25] Qu QJ, Duan H, Qiu XG, Huang KT. Research on interest management in large-scale virtual environment. Journal of System Simulation, 2002,14(4):485−487 (in Chinese with English abstract).

[26] Shi Y, Ling YX, JIN SY, Hu HP. Region matching algorithm based on hierarchical dividing of routing space. Journal of Software, 2001,12(5):758−767 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/12/758.htm

[27] Shi Y, Ling YX, Jin SY, Zhang CX. A hierarchical multicast address allocating strategy in data distribution management. Journal of Software, 2001,12(3):405−413 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/12/405.htm

[28] Zhou Z, Zhao QP. Study on RTI congestion control based on the layer of interest. Journal of Software, 2004,15(1):120−130 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/120.htm

[29] Reiher PL, Jefferson D. Virtual time based dynamic load management in the time warp operation system. Trans. of the Society for Computer Simulation Int'l, 1989,7(2):91−120.

[30] Glazer D, Tropper C. On process migration and load balancing in Time warp. IEEE Trans. on Parallel and Distributed Systems, 1993,4(3):318−327. [doi: 10.1109/71.210814]

[31] Jiang MR, Shieh SP, Liu CL. Dynamic load balancing in parallel simulation using time warp mechanism. In: Ni LM, ed. Proc. of the 1994 Int'l Conf. on Parallel and Distributed Systems. Washington: IEEE Press, 1994. 222−247.

[32] Cai WT, Turner SJ, Zhao H. A load management system for running HLA-based distributed simulations over the grid. In: Turner SJ, ed. Proc. of the 6th IEEE Workshop on Distributed Simulation and Real-Time Applications. Washington: IEEE Computer Society Press, 2002. 7−14.

[33] De Grande RE, Boukerche A. Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems. Journal of Parallel Distributed Computing, 2011,71(1):40−52. [doi: 10.1016/j.jpdc.2010.04.001]

[34] Robson ED, Azzedine B. Self-Adaptive dynamic load balancing for large-scale HLA-based simulations. In: Turner SJ, ed. Proc. of the 14th IEEE/ACM Symp. on Distributed Simulation and Real-Time Applications. Washington: IEEE Computer Society Press, 2010. 14−21. [doi: 10.1109/DS-RT.2010.11]

[35] Azzedine B, Robson ED. Dynamic load balancing using grid services for HLA-based simulations on large-scale distributed systems. In: Turnner SJ, ed. Proc. of the 13th IEEE/ACM Int'l Symp. on Distributed Simulation and Real Time Applications. Singapore: IEEE Computer Society Press, 2009. 175−183. [doi: 10.1109/DS-RT.2009.33]

[36] Azzedine B, Ahmad S, Zhang M. Efficient load balancing schemes for large-scale real-time HLA/RTI based distributed simulations. In: Roberts DJ, ed. Proc. of the 11th IEEE Symp. on Distributed Simulation and Real-Time Applications. Chania: IEEE Computer Society Press, 2007. 103−112. [doi: 10.1109/DS-RT.2007.25]

[37] Robson ED, Azzedine B. A dynamic, distributed, hierarchical load balancing for HLA-based simulations on large-scale environments. In: Ambra PD, ed. Proc. of the 16th Int'l Euro-Parallel Conf. Berlin, Heidelberg: Springer-Verlag, 2010. 242−253.

附中文参考文献:

[3] 龚建兴,韩超,邱晓刚,黄柯棣.构建可扩展的 HLA 联邦成员架构.系统仿真学报,2006,18(11):3126−3130.

[15] 黄健,郝建国,黄柯棣.基于 HLA 的分布仿真环境 KD-HLA 的研究与应用.系统仿真学报,2004,16(2):214−220.

[16] 刘步全,王怀民,姚益平.一种无死锁的时间管理算法.软件学报,2003,14(9):1515−1522. http://www.jos.org.cn/1000-9825/14/1515.htm

[17] 吕芳,周忠,魏晟,吴威.基于协调组的 RTI 时间管理方法.北京航空航天大学学报,2008,34(9):1084−1087.

[19] 苏年乐,李群,王维平.组件化仿真模型交互模式的并行化改造.系统工程与电子技术,2010,31(9):2015−2020.

[25] 曲庆军,段红,邱晓刚,黄柯棣.大规模虚拟环境中兴趣管理的研究.系统仿真学报,2002,14(4):485-487.

© 中国科学院软件研究所 http://www.jos.org.cn

[26] 史扬,凌云翔,金士尧,胡华平.基于路径空间层次划分的区域匹配算法.软件学报,2001,12(5):758−767. http://www.jos.org.cn/1000-9825/12/758.htm

[27] 史扬,凌云翔,金士尧,张晨曦.数据分发管理机制中层次化组播地址分配策略.软件学报,2001,12(3):405−413. http://www.jos.org.cn/1000-9825/12/405.htm

[28] 周忠,赵沁平.基于兴趣层次的 RTI 拥塞控制研究.软件学报,2004,15(1):120−130. http://www.jos.org.cn/1000-9825/15/120.htm

彭勇(1981－),男,广西贺州人,博士,讲师,主要研究领域为并行分布仿真,组合建模.

钟荣华(1985－),男,博士生,主要研究领域为系统建模仿真.

蔡楹(1984－),男,博士生,主要研究领域为面向服务的仿真,仿真资源管理与发现.

黄柯棣(1940－),男,教授,博士生导师,主要研究领域为系统仿真,控制理论与控制工程.

© 中国科学院软件研究所 http://www.jos.org.cn