基于模块化的移动 Agent 及其调度方法*

武成岗, 史忠植

(中国科学院 计算技术研究所 智能信息处理开放研究实验室,北京 100080)
E-mail: {wucg,shizz}@ics.ict.ac.cn
http://www.ict.ac.cn

摘要: 网络促进了信息的交流和数据的共享,人们正在研究和探索新的网络编程技术以使其更加有效地发挥作用.通过将执行代码从客户端传送至服务器的这种新的 Client/Server 程序执行思想已得到广大从事智能网络服务的研究和开发人员的认可,基于移动 Agent 的计算被视为该思想的一个代表.着眼于 Agent 的迁移机制的研究,为移动 Agent 设计了一个模块化结构模型,提出一种适合于 Agent 迁移过程实现和并发执行的调度方法,合理地运用模块复用的思想,减少了不必要的数据及代码迁移,缩短了移动 Agent 的平均执行时间.

关 键 词: 移动 Agent;迁移机制;功能模块调度序列;模块复用;并发执行
中图法分类号: TP18 文献标识码: A

近年来,计算机网络的规模以前所未有的速度得以发展.网络的发展促进了信息的交流和数据的共享,然而,其规模增长和形式多样性并没有直接带来计算能力的相应提高[1].目前,人们正在研究和探索新的网络编程技术,以使网络更加有效地发挥作用.通过将执行代码从客户端传送至服务器的这种新的 Client/Server 程序执行思想已得到广大从事智能网络服务的研究和开发人员的认可[2],基于移动 Agent 的计算被视为该思想的一个代表.从广义上来讲,移动 Agent 是计算机网络中的程序,它能够自动地从一个站点迁移到另一个站点,并代表其用户执行计算[3].

为了尽早地使移动 Agent 技术实用化,国内外学术界和企业界的研究机构在进行努力的探索,其代表工作有 IBM 的 Aglet[4],General Magic 公司的 Telescript[5],Mitsubishi 公司的 Concordia[6],Dartmouth 学院的 Agent TCL[7],Geneva 大学的 JavaSeal[1]和南京大学的 Mogent[8]等.然而移动 Agent 的技术目前尚不成熟,一些关键性技术尚需进一步研究,如 Agent 及其宿主的安全、身份验证、授权[3]、停靠机制等问题.其中迁移机制是移动 Agent 的核心技术之一[3,8].上述研究机构都对 Agent 的迁移方式作了一定的探索.Aglet,Telescript 和 Agent TCL 等将迁移条件和动作序列都隐含到 Agent 的代码中,虽然具有较强的迁移表达能力,但同时也造成了网络传输量大、Agent 设计者负担重的问题.Concordia 将迁移信息和 Agent 代码分离,用旅行计划(itinerary)来描述迁移过程,但其描述能力和灵活性还不够.Mogent 改进了旅行计划,将旅行计划和功能体完全分离,设计了一种较为灵活的过程级迁移方式.

然而,上述方法都是将 Agent 的代码作为一个整体进行迁移.经过分析我们发现,Agent 随着在不同的站点上的任务的不同,一般都分别执行 Agent 不同部分的代码.如果能对其代码进行合理分割或按功能进行模块化设计,有计划地调度代码中各个模块的去向,可以有效地减少数据的传输量,同时还可以使无数据相关的代码模块在各个站点上并发执行,从两个方面提高 Agent 的执行效率.为此,本文为移动 Agent 设计了模块化结构模型,提出了一种适合于 Agent 迁移过程实现和并发执行的调度方法,合理地运用模块复用思想,减少不必要的数据

* 收稿日期: 2000-11-27; 修改日期: 2001-03-01
基金项目: 国家自然科学基金资助项目(69790080);国家 863 高科技发展计划资助项目(863-306-ZT02-01-3)
作者简介: 武成岗(1969－),男,河南郑州人,博士,讲师,主要研究领域为人工智能,移动计算;史忠植(1941－),男,江苏宜兴人,博士,教授,博士生导师,主要研究领域为人工智能.

及代码迁移,缩短了移动 Agent 的平均执行时间.

本文第 1 节介绍移动 Agent 的模块化结构框架、Agent 构造服务器以及 Agent 的迁移方法.第 2 节介绍基于单机和局域网的移动 Agent 服务平台,着重讨论如何利用模块复用思想减少数据传输量、提高移动 Agent 的执行效率.第 3 节给出一个简单的实验示例.第 4 节介绍当前的相关工作.文章最后给出了结论.

1 移动 Agent 的模块化结构及其迁移策略

1.1 Agent的结构框架

为了尽可能地减少 Agent 迁移过程中的数据传输量,提高 Agent 设计的灵活性,本系统为移动 Agent 设计了如图 1 所示的模块化结构模型.

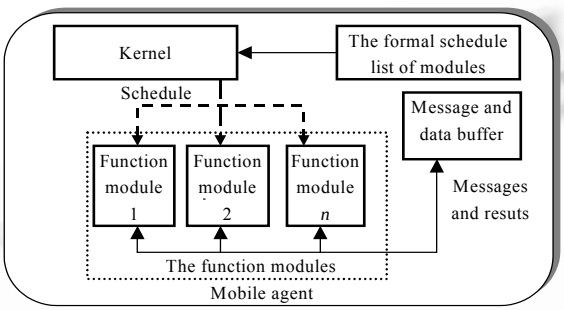

Fig.1 The framework of the mobile agent

图 1 移动 Agent 模块化结构模型的基本组成

Agent 由内核、功能模块调度序列的规范描述(下面简称调度序列)、消息及数据缓冲区和一系列功能模块组成.为了使移动 Agent 适合于各种平台,整个系统采用 Java 语言进行编写,其中每个功能模块作为一个线程来实现.

Agent 内核的作用是对调度序列进行分析,并根据它来调度各个功能模块(将在第 1.3 节中介绍).

功能模块调度序列是描述移动 Agent 工作过程的主体,由若干个段(segment)组成,其中每个段的基本组成如图 2 所示.

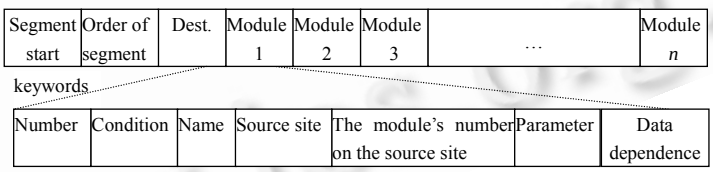

Fig.2 The structure of a segment

图 2 段的结构

一个段描述了 Agent 在一个站点上所应做的工作.每个段以一个保留字开始,紧接着是段序号,然后是 Agent 所应到达的目标站点,后面是 Agent 在该站点上应执行的若干个功能模块的描述.功能模块的描述分为模块序号、模块执行条件、模块名、模块的源站点、模块在源站点上的编号、初始参数、数据相关性.模块序号用于指示模块在该段中的序号;条件部分给出执行本模块所需的条件,只有该条件被满足,才启动此模块,否则跳过本模块;模块名给出功能模块的名称;模块的源站点和模块在源站点上的编号给出该功能模块的源代码所在的位置;后面是模块执行时需用到的初始参数;数据相关性用于描述本功能模块与哪些功能模块相关联,即该功能模块需用到哪些功能模块的执行结果,当存在结构相关时,该功能模块则需等到所需数据到达后才能启动执行.

图 3 是一个移动 Agent 的功能模块调度序列简单示例.

AgentID	Agent 10	**User's address**	UserIP	**AmountOfSegments**	3
Centification	UserId	Password			
SegmentStart	1	Workstation A			
Module 1	Null	Communiction	000001	AgentCreatingServer	Null	Null
Module 2	Null	Process_3D_1	000002	AgentCreatingServer	Input data	Null
SegmentStart	2	Workstation B			
Module 1	Null	Communiction	000001	AgentCreatingServer	Null	Null
Module 2	Null	Process_3D_2	000003	AgentCreatingServer	Input data	Null
SegmentStart	3	Workstation C			
Module 1	Null	Communiction	000001	AgentCreatingServer	Null	Null
Module 2	Null	Process_3D_3	000004	AgentCreatingServer	Input data	Null
Module 3	Null	ResultIntegreting	000005	AgentCreatingServer	Null	⟨1,2⟩⟨2,2⟩⟨3,2⟩

The bold is keywords

UserID and Password are used in authentication

AmountOfSegments is the amount of segments in this FSLM

⟨segment's number, module's number⟩

Fig.3 An example of the FSLM

图 3 功能模块调度序列示例

该功能模块调度序列规范描述的含义如下:Agent 生成之后,先迁移到站点 WorkStation A,执行功能模块 Communication,Process_3D_1;而后迁移到站点 Workstation B,执行功能模块 Communication,Process_3D_2;然后迁移到站点 Workstation C,执行功能模块 Communication,Process_3D_3 和 ResultIntegreting,由于模块间的相关性,模块 ResultIntegreting 在执行之前需等待站点 Segment 1 的第 2 个模块、Segment 2 的第 2 个模块、Segment 3 的第 2 个模块的执行结果的到来.在本例中,所有模块的代码均来自于站点 AgentCreatingServer 的功能模块库.

当某个功能模块执行完毕后,将其执行结果写入消息及数据缓冲区,供其他功能模块使用或作为 Agent 执行的最终结果传送给用户.当所有的功能模块按调度序列的要求执行完毕后,则表示该 Agent 执行完毕.

1.2 移动**Agent**构造服务器

为了使用户不必为繁琐的 Agent 编程所缠绕,系统设置了一个 Agent 构造服务器(如图 4 所示),协助用户自动地生成移动 Agent.该构造服务器包括功能模块库、移动 Agent 服务模块、调度序列记录表、Agent 登记表和 Agent 发射器以及图形用户界面(GUI).

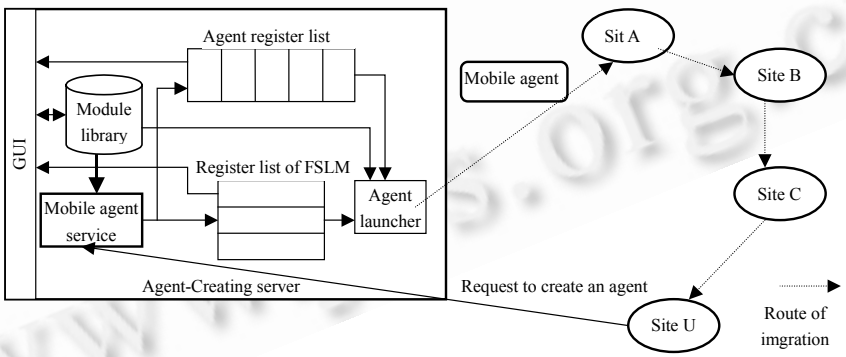

Fig.4 Agent-Creating server

图 4 移动 Agent 构造服务器

功能模块库是为了便于 Agent 设计而设置的,它为用户提供一些常用的功能模块.服务器管理员可以根据用户的需求,通过 GUI 来管理该模块库.

移动 Agent 服务模块的作用是接收来自于用户站点的请求,为用户生成 Agent.当站点 U 上的用户在完成一个调度序列的设计后,则可向移动 Agent 服务器发出一个请求原语,请求移动 Agent 服务器为其生成一个 Agent.请求原语格式如下:

Request-Create-Agent {

ServerIp IpAddress 1;

UserIp IpAddress 2;

UserTask TaskId;

ScheduleQueue schedule_queue;}

其中,ServerIP 后的参数给出服务器的 IP 地址,UserIP 后的参数给出用户的站点 IP 地址,UserTask 后的参数给出请求生成该 Agent 的任务 ID 号,ScheduleQueue 后的参数给出该 Agent 的调度序列.

服务模块收到该原语后,分析该请求中的调度序列是否存在错误.如果没有发现错误,服务模块将用户所设计的调度序列存入调度序列记录表中,并在 Agent 登记表中进行登记,等待 Agent 发射器按照调度序列将该 Agent 发往它所要去的站点(具体过程将在第 1.3 节中介绍),并发确认给用户.否则发错误警告给用户.

系统中还为服务器管理员设置了一个 GUI,供服务器管理员监测 Agent 登记表及调度序列记录表中的内容和管理功能模块库.

1.3 Agent的迁移和执行过程

在传统的方法中,Agent 一般要携带所有代码按照它的旅行路线依次在各个站点上移动.就上例而言,该 Agent 在每个站点上都有一部分代码不被使用,但在 Agent 移动过程中这些代码每次都要进行传输.另外,虽然在站点 Workstation A 上的运算结果仅在 Workstation C 上使用,但却要经 Workstation B 才能传到 Workstation C.再者,从调度序列中可以看出,除了 Segment 3 中的 ResultIntegreting 以外,其他功能模块均可以并发执行,但传统的方法中模块的执行只能串行进行.可见传统方法不但进行了不必要的数据传输,而且还影响了程序执行的并发性.

因此,我们采用下述方法来优化 Agent 的迁移过程.移动 Agent 发射器从 Agent 登记表中取出等待发送的 Agent 的序号,依据该序号从调度序列记录表中读取该 Agent 的调度序列.将 Agent 内核、调度序列发往各段所指示的目的站点.然后分析调度序列,如果 Agent 用到的功能模块在本服务器上,则从功能模块库中取出所需的模块并发往各段所指示的站点;如果所需的模块在其他站点上,则向该模块的源站点发出请求,请求对方将功能模块发往相应的站点.目标站点设有支持移动 Agent 的服务平台(将在第 2 节中介绍),该服务平台接收到每个数据后要发确认信息给移动 Agent 构造服务器.当发射器收到一个 Agent 的所有确认后,就将其从登记表中删除.

依上例,Agent 发射器作如下处理:将内核和功能模块调度序列规范描述广播到 3 个站点,将功能模块 Communication,Process_3D_1 送往 Site A,Communication,Process_3D_2 送往 Site B;Communication,Process_3D_3 和 ResultIntegreting 送往 Site C.

当目标站点的服务平台接收到某个 Agent 的内核后,则启动内核模块工作.

Agent 内核由两个线程组成,分别是 ModuleScheduling 和 ModuleListening(如图 5 所示).

线程 ModuleScheduling 的作用是:首先判断该 Agent 的调度序列是否已经到达该站点.如果尚未到达,则等待.若调度序列已经到达,则从调度序列中读取需启动的第 1 个模块名、执行条件及其相关关系.如果该模块的执行条件不满足,则跳过该模块.如果执行条件得以满足,检查该模块是否已经到达以及该模块是否等待别的某个功能模块的处理结果.当该模块的代码已经到达并且所需的数据也已准备好,则启动该模块工作,否则等待.一个模块启动后,再分析功能模块调度序列,重复上述过程,直至按调度序列的要求将各功能模块全部启动执行为止.

线程 ModuleListening 的作用是监测所有正在执行的模块.当某个功能模块执行完毕后,该线程按照调度序列中的数据相关关系的需要,将功能模块的执行结果发往相应的站点.

对于上述例子,在 Agent 整个工作过程中,只有在 Workstation C 上的 ResultIntegreting 模块具有数据相关性,需等待其他站点的数据.这样 Agent 内核一开始直接启动除 ResultIntegreting 以外的各个模块执行.当 ResultIntegreting 所需的数据到来后,再启动该模块执行.ResultIntegreting 执行完毕后,将最终结果传送给用户.

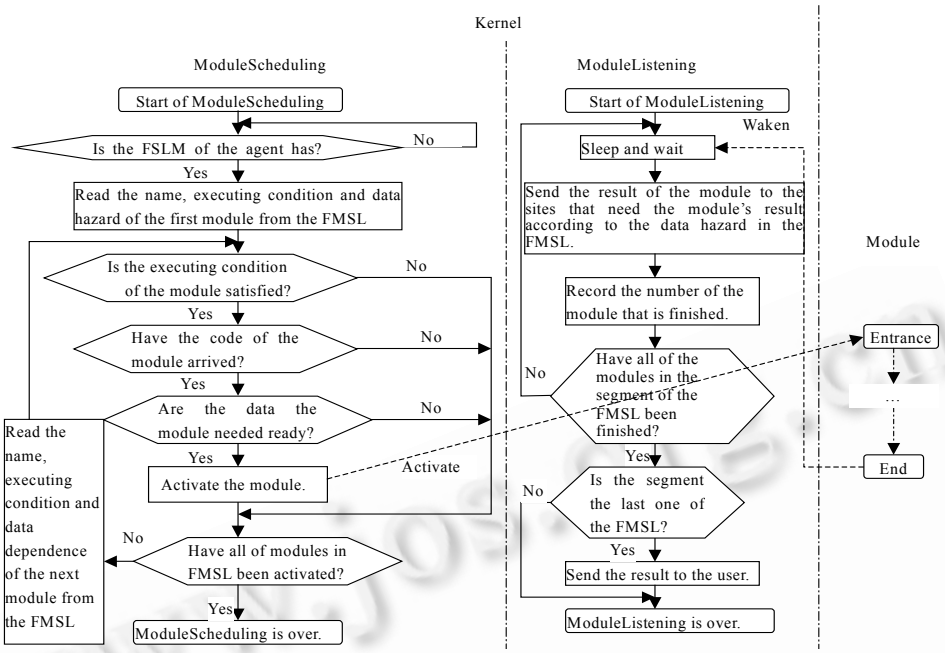

Fig.5 The flow chart of the agent kernel

图 5 移动 Agent 内核模块的基本流程图

2 移动 Agent 服务平台

2.1 宿主机上的Agent服务平台

为了支持 Agent 的迁移,我们在各个宿主机上设置了一个移动 Agent 的服务平台,该平台结构如图 6 所示.

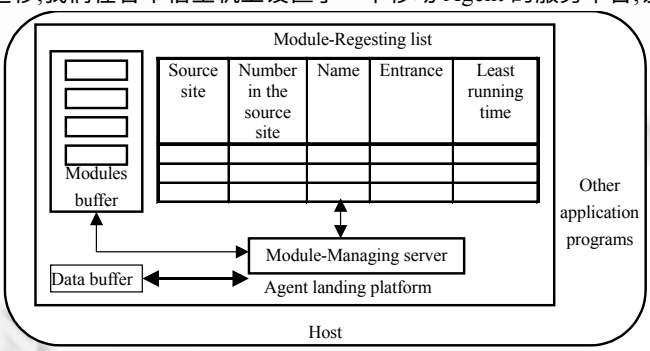

Fig.6 The serving platform in a single host

图 6 局域网中各站点 Agent 服务平台

移动 Agent 服务平台由如下几个部分组成:

- 功能模块管理服务程序:负责与远端主机联系,接收来自远端主机传送过来的 Agent 的各个模块并发确认信息;当接收到 Agent 内核后,则启动内核工作;另外,它还负责对功能模块缓冲区进行管理调度.
- 功能模块缓冲区:存储从远处传来的功能模块、Agent 内核和调度序列.为便于管理,系统将 Agent 内核和调度序列也作为功能模块来处理.
- 本机功能模块记录表:记录存储在功能模块缓冲区中的各个功能模块的源站点、在源站点上模块的编号、模块名、模块入口和最近一次被启动的时间.各个主机设计的功能模块在源主机内应有一个惟一的编号,因此,源站点和这个编号可以在网络中惟一地确定一个模块.最近一次启动时间的设置方法是:

当某个功能模块被系统启动执行时,用系统当前时间替换掉该域原有的时间.

- 数据缓冲区:在 Agent 工作过程中,各个功能模块将其执行的中间结果存储于该缓冲区,以备其他模块使用或作为结果传输到其他站点.

当 Agent 到达该站点时,服务平台上的功能模块缓冲区存储从远处到来的移动 Agent 的内核及其功能模块.当此 Agent 执行完毕,系统并不急于释放掉这些功能模块所占用的空间,而是暂存下来留作以后重复使用.

经分析,我们发现存在一些常用功能模块,如通信、KQML 解释器等.若对于某个领域的用户来讲,常用的功能模块将会更多.本系统的一个主要原则是丢掉那些不常用的模块,保留那些常用模块.功能模块管理服务程序采用最久未使用算法来调度缓冲区中的功能模块.每当新模块到来时,如果缓冲区中尚有空间可存储该模块,那么在空闲数据区存储该模块.若此时缓冲区已满,则删除缓冲区中最近一次被启动时间域中被使用最早的模块,为新的功能模块提供存储空间.

这里,Agent 的传输过程尚需进行一定的修改.当某个 Agent 准备要移到某台宿主机时,Agent 发射器首先与宿主机的功能模块管理服务程序交互,过滤掉那些宿主机上已有的功能模块,然后将剩余的功能模块发往宿主机.由于宿主机存储的是一些经常使用的功能模块,故重复利用率较高,减少了数据传输量,因此可以提高移动 Agent 的平均性能.

2.2 局域网中**Agent**服务平台的管理

一台机器的总空间十分有限,不可能提供很大的空间来存储备用功能模块.但局域网中的数据传输速率一般要比广域网快若干个数量级,因此若能有效地利用局域网资源,分布存储功能模块,从整体上进行调度,既可以解决缓存空间问题,也可以进一步缩短 Agent 执行周期.

在局域网中设置一个 Master 主机(一般由局域网中的某个服务器担当),其基本结构如图 7 所示.局域网中每个站点上的功能模块服务程序记录下 Master 的 IP 地址.Master 与局域网中其他主机的服务平台稍有不同,它的

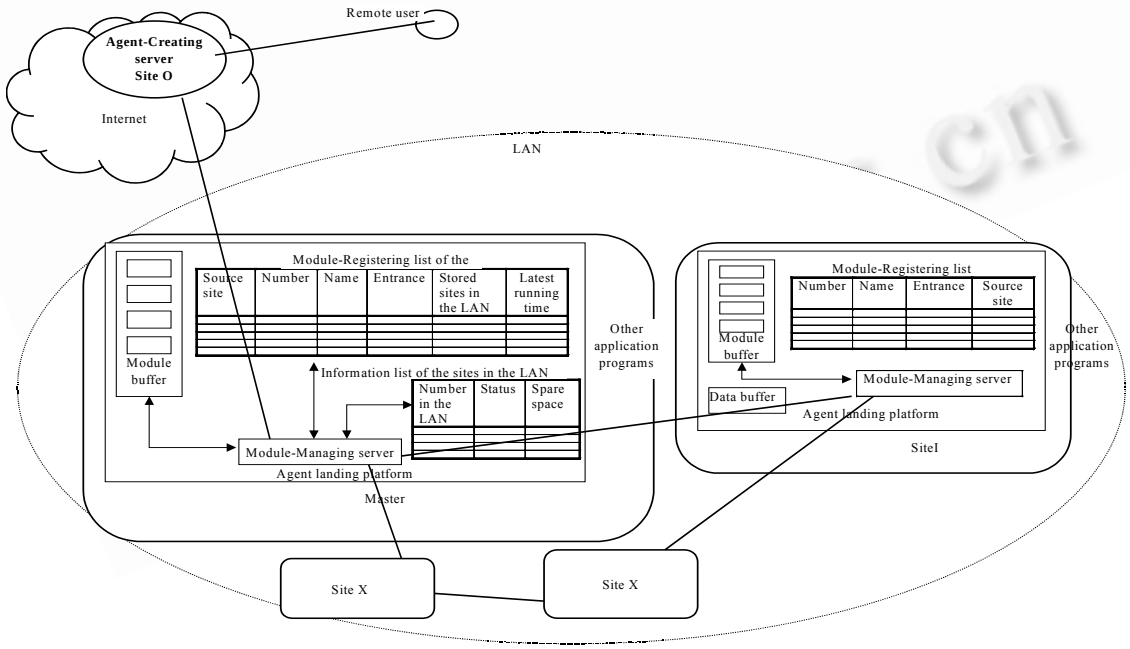

Fig.7 The management of the serving platforms in the LAN

图 7 局域网中各站点移动 Agent 服务平台

内部设立了一个局域网内功能模块记录表,该表记录包括本机在内的局域网内所有主机上现存功能模块的信息;另外,再设置一个局域网内各站点信息表,记录下局域网内各个主机的 IP 地址编号、状态(是否开机)以及各

站点功能模块缓冲区的剩余空间.Master 定时检测局域网的每台主机,判断局域网内每台主机是否处于开机状态,实时更新信息表中的状态字段.

当外部的某个移动 Agent 从局域网外的某个站点 Site O 要到达本网内的某个站点 Site I 时,Site I 将 Master 的 IP 地址发给 Site O,请 Site O 与 Master 联系.此时,Site O 便将该 Agent 所需模块的编号、源主机的 IP 地址以及模块大小形成一个列表 FMList,发送给 Master.Master 检查"局域网内功能模块记录表"和"局域网各站点信息表",判断在局域网内正在运行的主机上是否已有该 Agent 的部分模块,将已有并且可用的功能模块从 FMList 中滤掉.然后,Master 按照模块的大小统一安排 FMList 中剩余模块应存储在哪个站点.因为 Agent 要在 SiteI 上执行,所以 Master 首选 Site I 为主要的存储站点;对于剩下的功能模块,则另行安排到那些功能模块缓冲区尚有空余空间的其他主机;如果局域网内所有主机的所有缓冲区已满,则采用最久未使用算法替换掉整个局域网中最久未使用过的模块.Master 安排完毕后,形成一个安排结果 FMArrangeList,并发送给 Site O.Site O 按照 FMArrangeList 将功能模块发往局域网内相应的站点.局域网内的各个站点接收到 Site O 发来的模块后,修改本机的功能模块记录表,并发信息给 Master;Master 再根据该信息修改"局域网内功能模块记录表"和"局域网各站点信息表"中的对应项.这样,局域网相对于单机来讲,存储了更多的常用功能模块.

当 Agent 内核到达目的站点 Site I 后,Site I 上的服务平台则启动该内核工作.为了有效地利用这些功能模块,Agent 内核程序尚需作如下的调整.当 Agent 内核启动某个功能模块时,首先要查找本主机上是否已有该功能模块,如果已有,则执行;否则,向 Master 发出请求,询问本局域网内是否已有该模块,若有,则从相应的主机中将相应的模块取过来执行.若没有,则等待,直到该模块到达本局域网后,启动执行.

同样,某个功能模块启动执行时,服务程序应将启动时间发往 Master,由 Master 的服务程序将局域网内"功能模块记录表"中的内容进行相应的修改.由于 Master 记录下来整个局域网中各个模块的最近一次被启动的时间,其他主机便无须再记录这一项.

利用局域网中的各个站点的缓冲区,共同为移动 Agent 提供一个大的功能模块缓冲区.这样,可以存储更多的常用功能模块,进一步减少数据的传输,提高移动 Agent 的平均性能.

3 实验示例

本文采用一段三维动画制作程序作为模拟实验示例.整个三维动画中包括 3 个场景:第 1 个场景是室外景物,接下来切换到室内场景,最后是人物场景.

传统的制作方法大多是在单机上完成整个制作过程.这样要求该机器必须包括制作这 3 种场景的所有的素材库.另外,由于三维动画制作的数据运算量十分巨大,要想较快地完成整个制作,需要专用的高速工作站或高速计算机.

其实计算机网络中蕴含着极其丰富的资源,包含大量的软件、数据以及一些部分时间处于空闲状态的高速计算机和专用工作站.如果能对这些资源加以利用,可以使那些仅拥有低档计算机的用户也能在较短的时间内完成具有大运算量的任务.

对于上述示例,我们可以委派移动 Agent 在网络环境中利用网络资源来完成制作过程.图 8 是该示例所采用的模拟环境.

网络中有 3 个高速工作站(我们用 Pentium III 733/128M 内存微机来代替),分别是 WorkStation A,WorkStation B 和 WorkStation C,在它们上面分别存储着室外景物、房屋建筑和人物等 3 种素材库.其中 WorkStation B 和 WorkStation C 在同一个局域网中,网络速度是 10Mbps.其余机器均通过广域网连接,传输速度为 30kbps.用户 PC 和 Agent 构造服务器均采用较低配置的普通微机(Pentium 100/16M 内存).

针对上述网络结构和应用需求,我们设计了一个移动 Agent,从 Agent 构造服务器迁移到上述 3 个工作站去完成三维制作.相应的功能模块调度序列如图 8 所示.通信模块 Communication 负责 Agent 的段与段之间的通信;Process_3D_1,Process_3D_2 和 Process_3D_3 作用是利用当地的素材库对 3 种不同的场景进行处理;ResultIntegreting 的作用是综合 3 个段的执行结果.Agent 处理结束后,将结果传回 User 所在微机.

如果单独采用较低配置的微机处理完成上述任务,需运行时间 20 小时左右.单独采用高速工作站执行上述

动画制作,则需运行 2 小时 20 分.采用本系统进行模拟,由于移动 Agent 在 3 台高速服务器上并行执行各自场景的制作,因此完成整个任务仅需 1 小时 13 分.

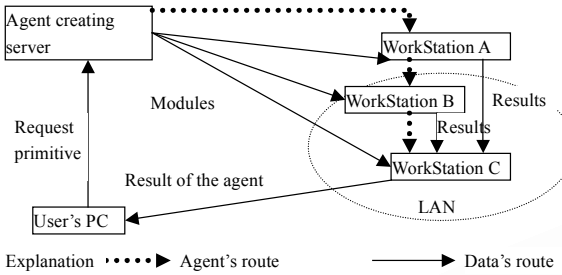

Fig.8 Test module

图 8 测试模型

4 相关工作

目前 Agent 的迁移常采用如下 3 种方法(见文献[3]):

- (正如前面所提到的)移动 Agent 携带其所有的代码,从一个站点移动到另一个站点.
- 一些系统不传送任何代码,而是将 Agent 所需的所有代码预装入各个宿主机上.
- 移动 Agent 在移动过程中不包括代码,而仅携带代码库的执行序列.在执行过程中,Agent 和存储代码库的服务器交互,从该服务器上下载所需的代码.

第 1 种方法显然会造成不必要的数据传输,并且不利于网络多机并行执行;第 2 种方法将所有的代码存储于宿主机,我们知道,程序代码各个部分的使用频度是不同的,将低使用率代码长时间存储于宿主机,将会占用大量的存储空间,这种方式对于大部分宿主机来讲是无法容忍的;第 3 种方式是在 Agent 执行过程中,用到哪个功能模块就从服务器下载相应的代码,该方法虽然能够减少代码数据的传输,但整个过程中代码的传输和 Agent 的执行是串行的,而且不进行代码复用,总的数据传输量仍高于本文所介绍的系统.另外,本系统采用模块化机制来构造移动 Agent,易于实现模块在各个站点的并发执行,从而缩短了 Agent 的整体运行时间.

5 结 论

本文为移动 Agent 设计了一个模块化结构模型、调度算法以及基于单机和局域网的模块复用方法.此方法减少了 Agent 移动过程中的数据传输量,合理地利用多机执行的并行性,从而缩短移动 Agent 的总的执行时间,改善了移动 Agent 的平均性能.因此,该思想能在一定程度上促进移动 Agent 技术进一步走向实用化.

当 Agent 用户较多时,构造服务器的负担会变得较重.在这种情况下,需通过增加构造服务器、提高通信带宽加以解决.这样显然会增加一些投资,但构造服务器的设置为众多 Agent 用户提供了方便,基本上免去了代码编写的负担,同时,整个系统在运作过程中可以提高移动 Agent 的并行性和执行效率,减少 Agent 迁移过程中的平均数据传输量.因此总的来讲,增加构造服务器的投资也是必要的.

本系统尚不适用于那些在运行过程中需根据具体情况实时选择功能模块、动态改变调度序列的移动 Agent.在 Agent 宿主机上采用最久未使用算法来管理功能模块缓冲区的空间并不是最佳的方法.针对这些问题,我们正在做进一步的研究和探索.

References:

[1] Wei, Jun, Feng, Yu-lin. Analysis of formal models and methods on mobile computing. Journal of Computer Research and Development, 2000,37(2):129~139 (in Chinese).

[2] Harrison, C.G., Chess, D.M., Kershenbaum, A. Mobile agent: are they a good idea. Technical Report, IBM Research Division, T.J. Watson Research Center,1995. http://www.research.ibm.com/massdist/mobag.ps.

[3] Karnik, N.M.,Tripathi, A.R. Design issues in mobile-agent programming system. IEEE Concurrency, 1998,6(3):52~61.

[4] Karjoth, G., Lange, D.B., Oshima, M. A security model for aglets. IEEE Internet Computing, 1997,2(4):68~77.

[5] James, E. White: mobile agents. Technical Report, Los Angles; General Magic, 1995.

[6] Wong, D., Paciorek, N., Walsh, T., *et al*. Concordia: an infrastructure for collaborating mobile agents. In: Proceedings of the Mobile Agents: 1st International Workshop. Lecture Notes in Computer Science 1219, Berlin: Springer-Verlag, 1997.

[7] Kotz, D., Gray, R., Nog, S., *et al*. AGENT TCL: targeting the needs of mobile computers. IEEE Internet Computing, 1997,1(4):58~67.

[8] Tao, Xian-ping, Lǔ, Jian, Zhang, Guan-qun, *et al*. Design and implementation of a mobile agent structured migration mechanism. Journal of Software, 2000,11(7):918~923 (in Chinese).

附中文参考文献:

[1] 魏峻,冯玉琳.移动计算形式理论分析与研究.计算机研究与发展,2000,37(2):129~139.

[8] 陶先平,吕建,张冠群,等.一种移动结构化迁移机制的设计和实现.软件学报,2000,11(7):918~923.

Module-Based Mobile Agent and Its Schedule Method*

WU Cheng-gang, SHI Zhong-zhi

(*Laboratory of Intelligent Information Processing*, *Institute of Computing Technology*, *The Chinese Academy of Sciences*, *Beijing* 100080, *China*)

E-mail: {wucg,shizz}@ics.ict.ac.cn

http://www.ict.ac.cn

Abstract: The development of network has greatly promoted the information communication and the data sharing. Many people are trying their best to develop new programming technologies to utilize the network more efficiently. The idea of performing client-server computing by transmission of executable programs between clients and servers has become highly popular among researchers and developers who are engaged in intelligent network services. Computing based on mobile agents is an important aspect of this idea. This paper focuses on the research of the migration process of agents. A model based on modules is devised for constructing agents. A concurrent schedule method is presented, by which the agent migration can be easily implemented. Most of the unnecessary transmission of codes and data can be avoided by module reuse. Consequently, the executing period of mobile agents is reduced and their efficiency is improved.

Key words: mobile agent; migration process; schedule list of module; module reuse; concurrently execute

* Received November 27, 2000; accepted March 1, 2001

Supported by the National Natural Science Foundation of China under Grant No.697908001; the National High Technology Development 863 Program of China under Grant No.863-306-ZT02-01-3