计算机科学 2002Vol. 29№. 10

基于元组空间的 Agent 协作模型的分析与实现*)

The Analysis and Implementation of Tuple Space Based Agent Coordination

郭 中 王惠芳 黄永忠 郭金庚

(解放军信息工程大学 1001 信箱 郑州 450002)

Abstract The coordination of multi-Agent is an important issue in multi-Agent system application. The paper analyses the two widely used coordination ways which are direct coordination and indirect coordination and proposes a paradigm combined with the above two coordination ways. The paradigm is implemented with the help of Purdue Bond Agent Framework and successfully run by a distributed computation example. At last the recent tendency on tuple space is introduced.

Keywords Tuple, Tuple space, Coordination, Message-passing, Agent

一、引言

在基于 Agent 的分布式处理系统中,由于单个 Agent 不能充分利用系统中的资源,所以一个复杂的任务常常分解为多个简单的任务,每个简单的任务由一个 Agent 完成,通过多个 Agent 的协调合作共同解决问题,故 Agent 之间如何很好地协作是多 Agent 系统设计中一个很重要的方面。

在层出不穷的 Agent 系统中,Agent 之间的合作方式不外乎两种:直接合作和间接合作,直接合作建立在 Agent 之间直接通信的基础上,而间接合作则通过第三方作为媒介。目前,大多数 Agent 系统都采用基于消息传递的直接合作方式,消息传递要求通信前必须事先知道对方的地址、名字,当 Agent 移动时,其地址会经常发生变化,所以为支持直接通信,Agent 系统除了要在每台机器安装 Agent 运行平台外,还要提供名字服务(为 Agent 分配一个唯一的名字)、定位服务(确定目前 Agent 的地址)、消息缓冲(保证当 Agent 在移动过程中也能接收消息)等辅助设施,并要对 Agent 通信语言、通信协议的语法、语义进行分析以做出相应的动作,这种合作方式虽然很灵活,但增加了设计 Agent 应用程序的复杂度。

间接合作以一个全局共享的网络缓冲区(典型的代表是元组空间)作桥梁,不管 Agent 移动到何地,只需把要交换的信息内容组织成有序的域,例如("股票价格",300.0)是二个域的元组(tuple),并把它放进元组空间(tuple space),至于哪个 Agent 使用、何时使用、怎样使用都不关心,对该元组感兴趣的 Agent 形成动态的接收方集合,不必知道对方的地点、名字,只要获取元组并处理即可。对特定元组的阻塞读自然而然形成了 Agent 之间的同步协调机制,这种简单的合作方式尤其适合移动 Agent 和动态变化的环境,不要求全局性的名字服务、定位服务等设施,使 Agent 之间的协作由读/写元组、阻塞读等简单的操作就能进行,大大简化了 Agent 应用程序的设计。

一般来说,Agent 系统的交互有三种类型:Agent 平台之间的交互、Agent 与 Agent 平台之间的交互、Agent 之间的交互,笔者认为前两种适宜采用直接通信方式,后一种采用间接合作模式,下文将详细阐述这种构思,接着介绍在这种模型上实现的一个分布式计算的应用实例,然后给出实现细节,最后进一步探讨未来的发展趋势。

二、直接通信与间接通信相结合的 Agent 系统构架

Agent 平台之间在下列情况下会出现交互:①创建远程 Agent;②Agent 的移动。这些都要求 Agent 平台之间首先建立网络连接,再把创建或恢复 Agent 的数据传输到目的地平台,一旦创建或恢复 Agent 成功后再释放连接。这一过程采用直接通信的方式效率高,因为远程创建和移动的目的地一旦决定下来就是很明确的,如果采用间接通信的方式把创建或恢复 Agent 的有关数据放进元组空间,则可能被其它 Agent 平台取走,使 Agent 创建或移动到不应该去的地方。

Agent 与 Agent 平台之间的交互非常频繁,如 Agent 的启动与停止、存取本地资源完成任务等都要与 Agent 运行平台打交道,这属于同一台机器上不同进程间的交互,用消息传递直截了当,没有必要通过第三方介入。但在 Agent 应用领域,Agent 之间的交互随着不同种类的应用有不同的需求,元组空间的引入能使合作以匿名、不要求时间和地点的方式进行。

综上所述,我们设计的 Agent 系统组成结构很简单:Agent 平台+元组空间服务器(Tuple Space Server)。

三、基于上述框架的分布式计算模型

在分布式计算中,关键是计算任务的分解和中间结果的处理,若采用元组空间模型则能很好处理,下面结合一个例子加以详细说明:

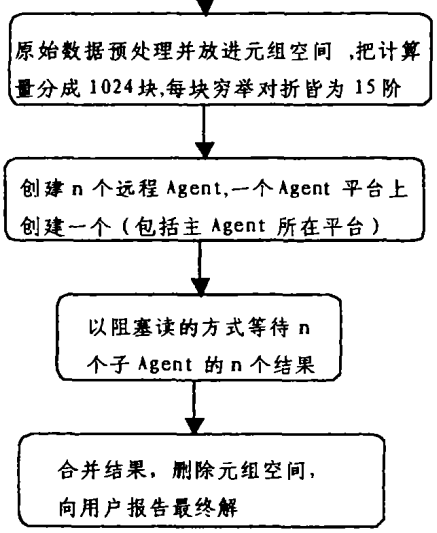
由于工作需要,我们要用哈达姆变换解 40 元、优势为 75%、10000 个方程的含错方程组,经过理论上的推算,最佳对折阶数是 15 阶,则穷举阶数是 25 阶。首先,在本地创建一个 Agent 担任主 Agent,主 Agent 在元组空间服务器上申请一块元组空间,把原始参数如方程组、可用的机器数等放进元组空间,并把穷举 25 阶的计算量分解成 2^{10}块(每块的计算量包括穷举 15 阶、对折 15 阶),以("F",0)、("F",1),…,("F",

*)该课题受到军队攀登工程项目资助,编号为 00230。郭 中 博士生,主要研究方向为分布式对象处理、多 Agent 系统。王惠芳 博士生,主要研究方向为多 Agent 系统、信息安全。黄永忠 讲师,主要研究方向为分布式对象处理、软件工程。郭金庚 教授,博士生导师,主要研究方向为分布式对象处理、多 Agent 系统、信息安全。

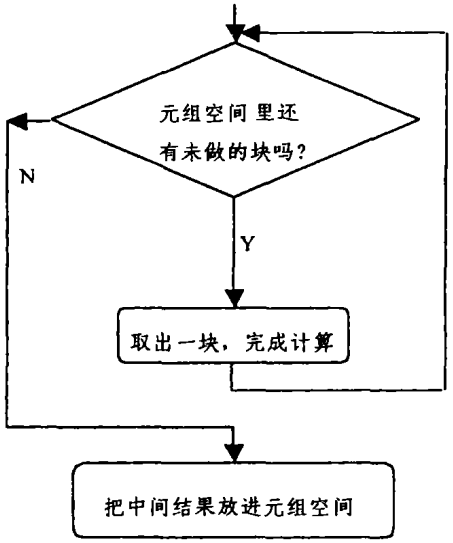
1023)的形式放进元组空间,表明整个的计算任务有1024块,每块都还未计算;第二步,主Agent根据当前可用的机器数,在每个Agent平台上(包括主Agent所在平台)创建一个子Agent,创建成功后,启动子Agent运行,自己则以阻塞读的方式等待所有子Agent以元组形式递交的中间结果,当所有子Agent把中间结果写进元组空间时,主Agent被唤醒,合并中间结果,得到最终解返回给用户。对子Agent来说,首先从元组空间中读出原始参数,再以("F",Integer)为模板查询是否还有未做完的块,若有则取出,调用相应的算法完成计算,循环此过程直至取不到块为止,此时子Agent把计算结果放进元组空间,并将自己杀死。

采用这种模型,任务的分配是动态的,在档次高的机器上运行的Agent取的块多,算得也多,Agent之间仅以读/写元组的简单形式协调,用户可把更多的精力放在优化解方程组的算法上,不必过多地关心合作问题。另外,如果解方程组的算法是用其它语言如C编写的,则该模型也很容易对其包装。

主Agent流程图



子Agent流程图



四、实现技术

我们的研究工作是在美国Purdue大学计算机系在网上发布的公开代码软件Bond Agent Framework(based Java)作为原型的基础上进行的,Bond Agent Framework以面向对象的技术、灵活的集成框架实现了BDI模型,我们对其进行了详细分析,删除了许多冗余复杂的成份如Directory Server、Persistent Storage Server、Authentication Server、Distributed Awareness、Blueprint等,保留了核心精华部分,例如Agent平台之间的消息通信、Agent的远程创建等,并增加和调整了许多模块,通过这种设计改造不仅精减了代码空间而且使结构更清晰可用。另外,Bond系统集成了IBM TSpaces技术,把元组、元组空间、元组空间服务器皆作为Java对象来处理并提供了API如read、write、WaitToRead、WaitToTake等对元组空间进行操作,十分方便。

首先,在局域网里可用的机器上装入裁减后的Bond Agent Framework作为Agent运行平台,然后,任选一个Agent平台在其控制面板上按下启动元组空间的按钮,则会生成bondTupleSpace类的一个实例,在它的构造方法中,创建TSServer守护线程,在8200端口侦听其它客户发来的操作元组空间的请求。在主、子Agent的模型里,标明当前元组空间服务器的IP地址和元组空间的名字即可。

实验数据表明,在我们最快的一台联想开天4600(P4,内存128M)上完成上例中的计算需5.7天,而在一台IBM(P2,内存128M)上则需21.3天,采用基于元组空间的分布式计算模型,在4台机器(两台联想开天4600 P4、两台IBM P2)上费时52.4小时,9台机器上(三台联想开天4600 P4、六台IBM P2)花了32.5小时,速度得到了较大提高,Agent充分利用了网络中的空闲CPU资源,为用户完成了任务。

需要说明的是,虽然元组空间服务器可创建多个元组空间,用于多个应用程序,但由于元组空间是集中处理模式,为避免单点失效引起的负效应,最好在一个元组空间服务器上仅创建一个元组空间用于一种应用程序,其它种类的应用可在别的Agent平台另起动一个元组空间服务器,这样互不影响,便于保证在同一个Agent系统中不同任务的顺利完成。

五、进一步讨论

Agent以基于元组空间的模型进行协作,并不是说Agent之间不能进行直接协调,例如在Bond Agent Framework中,Agent之间能对等地以XML或KQML作为通信语言、"Agent Control"作为通信协议、处理get-state、start-agent、stop-agent、kill-agent、getModel、setModel、populate-model等常用的消息内容,当要解决的问题中仅存在唯一解时,不管是哪个子Agent先做出来的,只要把结果放在元组空间中,就会触发主Agent向其它子Agent发kill-agent消息,将所有正在运行的子Agent杀死。元组空间技术只不过未给已有的Agent系统引入其它更多的与应用程序有关的交互协议。

最近,国外几家最新开发的Agent系统如MARS、TuCSoN等提出了可编程反应的元组空间(Programmable Reactive Tuple Space)的概念,其核心思想是除了支持通常的元组空间操作外,还能预先对元组空间的行为编程即当读写指定的元组操作发生时,元组空间能作出预定的反应,而本文用到的元组空间是被动的元组空间,它仅支持元组的读写,当读/写/删除元组的操作发生时无任何响应,应用程序需要自己去读取元组再进行处理,即使有事件通知服务如IBM TSpaces

(下转第51页)

了选出正确信息的工作，因此移动 agent 的自治性没有很好地体现出来。

由于插入了干扰码，移动 agent 本身的性能相应降低。假设在移动 agent 的 N 个相互独立的意图上，分别添加 M－1 组干扰码，则移动 agent 在这些意图上花费的时间为原来时间的 M 倍。因此移动 agent 性能的降低可表示为：

相互独立的意图数目×插入的干扰码数目，即 N×M。移动 agent 性能的降低主要体现在计算、通信和数据传输上。

4.2 意图收缩分析

假设移动 agent 被派出收集一些项目的信息，并且用一个评估函数来评估这些信息。这个评估函数能通过对各个属性进行加权，找到这些被访问主机中的最优者。评估函数为：$F(x_1,x_2,\cdots,x_{n-1},x_n)=a_1f_1(x_1)+a_2f_2(x_2)+\cdots+a_{n-1}f_{n-1}(x_{n-1})+a_nf_n(x_n)$。

其中，变量 x_i 表示移动 agent 的意图 x_i，a_i 表示意图 x_i 在移动 agent 的决策中所占的权值，且 a_i 满足条件：$\sum_{1\leqslant i\leqslant n}a_i=1$。

可通过对评估函数的分裂实现对 agent 的分裂。如下所示：

$$F(x_1,x_2,\cdots,x_{n-1},x_n)=\sum_{1\leqslant j\leqslant M}F_j(x_1,x_2,\cdots,x_{n-1},x_n)$$

$$a_i=\sum_{1\leqslant j\leqslant M}a_{j,i}$$

$$F_j(x_1,x_2,\cdots,x_{n-1},x_n)=\sum_{1\leqslant i\leqslant M}a_{j,i}f(x_i)$$

将函数 F()分裂后得到的函数 F_i()代替原移动 agent 中的函数 F()。

假设移动 agent 中有评估函数：

F()= 0.2(price)+ 0.3(weight)+0.2(battery-life)+0.3(additional features)

通过意图分裂，可将此原 agent 分裂为 3 个相互独立的 agent，每个 agent 分别有评估函数 F_1()，F_2()，F_3()：

F_1()=0.3(price)+0(weight)－0.1(battery-life)+0.2(additional features)

F_2()=0(price)－0.1(weight)+0(battery-life)+0.1(additional features)

F_3()=－0.1(price)+0.4(weight)+0.3(battery-life)+0(additional features)

F()=F_1()+F_2()+F_3()

由于 agent 在不同的时间到达远程主机，远程主机仅能对它们进行单个的分析，而不能把它们联系起来得到原始的求和因子，因此即使远程主机分析了单个 agent 的评估函数，但它得到的评估算法也是不完全的。如果远程主机要修改 agent 收集的信息，那么除非对这 3 个看来相互独立的 agent 进行的修改是完全相同的，否则，当所有的 agent 返回其主机后，主机会发现在这个远程主机上获得的信息是不可信的。

假设有 t 个原 agent，每个 agent 分裂为 M_i 个 agent(i=1,2,…,t)，即总共有 L 个 agent：

$L=M_1+M_2+\cdots+M_t$

在 L 个 agent 中，主机分辨出其中 M_i 个 agent 为相关 agent 的概率 p(i)为，$p(i)=1/C_L^{M_i}$。当 $M_1=M_2=\cdots=M_L=L/2$ 时，p(i)达到最大值。

与意图扩展法相同，由于原移动 agent 的分裂，移动 agent 本身的性能也相应降低。假设原 agent 中有 K 个意图，原 agent 分裂为 M 个相互独立的 agent，每个 agent 中也带有 K 个意图，则移动 agent 性能的降低为：K×M。

小结 如何保护移动 agent 免受恶意主机的攻击，是移动 agent 安全问题中最困难的部分。在本文中，采用基于意图的方法来保护移动 agent 数据机密性，根据概率学的方法证明能起到一定的作用。但这种方法也存在着不足之处：(1) 必须要以牺牲移动 agent 的部分性能为代价；(2)当有多个恶意主机同时对移动 agent 进行攻击时，主机获得移动 agent 意图的概率就会大大增加。我们今后的工作将围绕如何解决以上问题展开。

参 考 文 献

1 Sander T, Tschudin C F. Protecting Mobile Agents Against Malicious Hosts, in G. Vigna(ed.), Mobile Agents and Security, 1998

2 Sau-koon NG, Kwok-wai Cheung. Intention Spreading: an Extensible Theme to Protect Mobile Agents from Read Attack Hoisted by Malicious Hosts, in Jiming Liu, Ning Zhong(ed.), Intellingent Agent Technology: System, Methodologies and Tools, World Scientific, 1999. 406～415

3 Hohl F. A Protocol to Detect Malicious Hosts Attacks by Using Reference States, 1999

4 董红斌，石纯一．移动 agent 系统的安全问题．计算机科学，2000，27

5 罗飞，邱玉辉．基于意图的移动 Agent 安全机制研究．中国人工智能进展 2001．见：中国人工智能学会第 9 届全国学术年会论文集，2001，12

(上接第 29 页)

中的 eventRegister，也只是当对感兴趣的元组操作发生时(如写、删除)调用客户端的 Callback 方法，在 Callback 方法中客户可对取到的元组进行处理。

可编程反应的元组空间使元组空间表现出一定的智能行为，对提高多 Agent 系统的适应性、健壮性大有益处，例如对移动 Agent 来说，一个关键问题是确定它应该移到哪台主机，一种可行的方案如下：在每个 Agent 平台创建一个固定 Agent，它每隔半小时向元组空间更新元组，元组的内容可以是当前的 CPU 利用率、CPU 速度、内存资源等，利用元组空间服务器端的 Reaction 方法(该方法是在 MARS 系统中的元组空间软件包中定义)可把元组空间的反应编程为：若该平台的负载过多、资源紧张时，查询剩余的 Agent 平台看是否有负载轻的，若有则发出使 Agent 移动的消息，若发现有的平台隔半小时还没有更新元组，则可认为该平台已崩溃，可在别的平台上再启动一个 Agent，不仅使 Agent 的设计轻松简单，还增强了对 Agent 平台的监控能力。如何把可编程反应元组空间技术融合进我们现有的 Agent 系统中是进一步研究的问题。

参 考 文 献

1 Ricci A, Denti E, Omicini A. Agent Coordation Infrastructure for Virtual Enterprises and Workflow Management. Cooperative Information Agents Ⅴ, 5th International Workshop(CIA 2001), Modena (Italy), September 2001, Proceedings, LNCS 2182, Springer-Verlag, 2001

2 Ricci A, Omicini A, Denti E. Enlightened Agents in TuCSoN. Daqli oqqetti aqli agenti: tendenze evolutive dei sistemi software, AIA * IA/TABOO Joint Workshop(WOA 2001), Modena (I), Sep. 2001

3 Lehman T J, Cozzi A, Xiòng Yuhong. Hitting the distributed computing sweet spot with Tspaces. ELSEVIER Computer Networks, 2001, 35: 457～472

4 Boloni L, Jun K. The Bond Agent System and Applications. March 2, 2000. http: // bond. cs. purdue. edu/publication/ASAMA. PS