基于移动 Agent 的构件运行支撑研究与实现*)

杨 玫 胡海洋 陶先平 吕 建

(南京大学计算机软件新技术国家重点实验室 计算机软件研究所 南京 210093)

Design and Implementation of Runtime Support for Mobile-Agent-Based Component Software

YANG Mei HU Hai-Yang TAO Xian-Ping LU Jian

(State Key Lab. for Novel Software Technology, Institute of Computer Software, Nanjing University, Nanjing 210093)

Abstract Currently, component software is getting more and more widely used, but the deficiencies in traditional wiring mechanisms may, to some extent, constrain the application of component technology in network environment. After analyzing these deficiencies, the authors propose a new smart wiring mechanism based on mobile agent. This mechanism achieves the following advantages: 1)flexible later assembly mechanism; 2)reduction of network dependency; 3)less network latency and improved communication efficiency.

Keywords Mobile agent, Grouptable, CORBA, Interaction

1. 引言

分布式环境中,应该采用某种连线将处于不同地址空间的构件组装起来以形成一个完整的系统。因此,作为各个构件之间"粘合剂"的连线机制极其重要。基于开放系统互连参考模型[1],构件间的连线机制可以分为三级:(1)低级的连线机制,例如 Socket[2],它只提供了基本的传输语义,其优势在于几乎任何一台具有网络连接的机器都支持 Socket 接口,并且用它编写应用花费低,效率高。其缺点在于要求程序员关注较多的构件之间交互的细节,而这些细节与逻辑应用无关。(2)高级的连线机制,例如 Erlang[3],它通常与某一特定的编程环境结合在一起,只能使用某种特定的编程语言。程序员要遵从这样一种新的程序设计语言与风范,而用其编写的应用程序很难与其他的语言进行互操作。(3)中间件方法[4],介于上述两种方法之间,例如 CORBA[5]、DCOM[6,7]、JavaRMI[8]等。它们对网络进行了适当的抽象,并提供了丰富的服务,使得用户感觉不到系统的服务和外部服务器的服务的区别,这种透明性是中间件方法的优势所在。

通过对以上几种方法进行分析,能够发现它们都存在以下缺点:(1)缺乏灵活性。只能传输数据和控制,而代码和过程的状态不能传送;(2)效率低。其主要原因在于完成一次任务需要多次的远程通信。考虑到现在有限的网络带宽,这样将使系统效率降低;(3)坚定性差。在远程方法调用的整个过程,这些方法都要求调用构件和被调用构件之间的网络不能断开,这样建立起来的应用在很大程度上依赖于当前十分脆弱的网络;(4)装配欠灵活。构件的功能和其结构机制往往融合在一起,于是构件结构的任何改变均会导致整个构件的重写,尽管其功能部分与从前一样,因此难以对灵活的装配手段提供有效支持。这些不足在某种意义下制约了构件技术在网络环境下的应用[9]。

为了探索解决上述各类问题的方法和途径,我们提出了一种基于移动 Agent 技术和方法的 Cogent 构件框架,将构件功能和构件结构两部分加以分离,然后用基于 Agent 的"智能连线"来取代传统的基于 RPC 的连线,从而克服了传统连线机制的灵活性差、效率不高和坚定性不强的问题。

2. Cogent 构件框架

2.1 构件结构

在 Agent 环境中,每个构件由五部分组成:构件体、请求接口(request interface)、服务接口(service interface)、组调用表(GROUP TABLE)、定位表(LOCATION TABLE)。功能体主要是该构件提供的服务,它可以由多种语言书写。为了实现基于 Java 的智能总线和异构的构件功能体的连接,从而支持跨平台的构件的相互操作,我们采用一种类似于 COBRA IDL 的中性语言来描述构件的结构(包括请求接口和服务接口)。为了给构件的组装提供较大的灵活性,智能连线分成两部分,其一是组调用表,它是基于其他构件所提供的接口,可采用各种形式对功能体中的各种方法在组装时刻加以灵活解释,其二是定位表,它提供了组调用表中所涉及到的各个构件在网络中的位置信息。实际上,组调用表和定位表这两部分合在一起即可形成一个特定的移动 Agent,此 Agent 通过流动将构件的一系列相关的调用请求传给相应的服务构件执行,执行完成后返回结果。

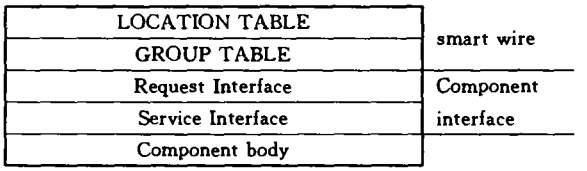

图 1 component structure

由于构件通常是由软件生产商独立地开发的,它们通常

*)本文受九五科技攻关项目(96-729-1-08)、863 高科技项目(2001AA113110)、国家自然科学基金项目(69873021)和国家杰出青年基金项目(61525204)资助。杨 玫 硕士,主要研究领域为构件技术、软件过程技术。胡海洋 主要研究领域为构件技术、软件过程技术。陶先平 博士,副教授,主要研究领域为对象技术,移动 agent 技术。吕 建 博士,教授,博士导师,主要研究领域为形式化方法,对象技术,分布计算技术和构件技术。

是用不同的语言书写的，运行于不同的系统平台上。在Cogent系统中，这些构件之间是通过移动Agent组装起来，使得它们能够相互对话，从而形成一个紧密联系的软件系统。

如图2所示，移动Agent构成了Cogent的连线机制，Agent用Java这样一种"世界语"的程序设计语言书写。因此，任何一台安装了Java虚拟机的机器都将支持Java语言的解释执行，从而平台的异构性为Java虚拟机所屏蔽。处于不同平台上运行的构件可以通过中介Agent来相互对话。另一方面，OMG已经定义了IDL及IDL到多种实现级程序设计语言的映射，且OMG IDL已经成为ISO国际标准。因此，Cogent使用IDL作为中间语言，实现了Java语言与其他语言的翻译。这一过程可以简单地表述为：语言A⇔OMG IDL⇔Java⇔OMG IDL⇔语言B。

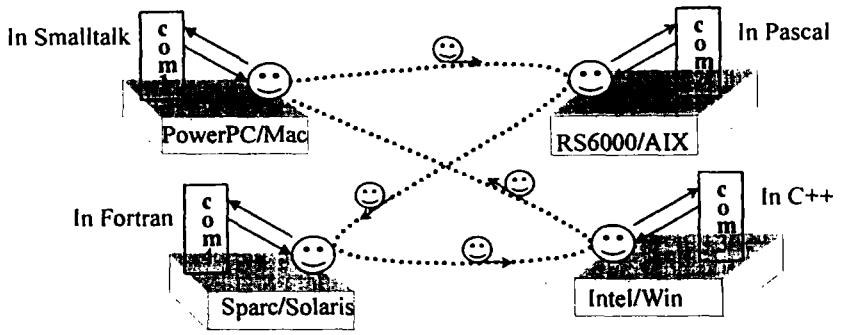

图2 mobile Agent wire

2.2 Cogent的智能连线

Cogent的智能连线包括组调用表和定位表两部分。

组调用表：组调用表在智能连线中起主导作用，它给出了整个控制流。具体说来，一个组调用表将一些关系密切的方法调用和方法后处理组织在一起。

图3显示了组调用表的一般结构。组调用表所有的程序代码和数据定义必须与Java语言中的一致。Group_name(param_list)形成了组调用表的签名，参数表中每一个参数用一个三元组(mode,type,name)来表示；Mode指出参数传递的方向有三个可能的选择：IN、OUT或INOUT；Private data用来定义新的类型、临时变量和命名常量；Type指出该组调用表是哪种类型，有四种选择SEQ、SLOOP、PAR或PLOOP；Condition给出了循环的结束条件，只有在类型为SLOOP或PLOOP时才有意义；Component reference指一个构件的具体实例；I.M()中I指出方法m属于哪个接口，()中的参数为实参；Post-processing是一小段Java代码，执行方法调用后的必要工作。

Group_name(…parameters…)				
Group Private Data				
TYPE	[condition]	Component reference	I.M(…)	Post-processing

图3 Group Table structure

Location_Table_name			
R1	Host1	Component11	Inst1
R2	Host1	Component12	Inst2
R3	Host2	Component21	Inst3
…	…	…	…
…	…	…	…

图4 Location Tabic Structure

定位表：定位表用于描述构件软件系统所需的构件(包括构件对象的初值)所在的位置信息。移动Agent正是根据定位表的信息来决定流动的目标节点。定位表的结构如图4。其含义是，定位表中的R1的初值为节点Host1上的构件Component11的实例Inst1，R2的初值为节点Host1上的构件Component12的实例Inst2等等，依次类推。若Ri初值为空，则表示Ri在定位表中将由方法后处理来给它赋初值。

3. Cogent运行支撑环境

3.1 Cogent构件运行时的交互

Cogent运行时的交互过程可划分为两部分：客户端交互和服务器端交互，Agent在网上的迁移作为两部分间联系的纽带。

组调用表和定位表的信息经过翻译器的作用，产生用C和Java书写的智能线源代码文件。输出依据其功能可分为三个部分：Marshaller，Agent管理器和Agent。Marshaller用C书写，如其名字所示它扮演了构件功能体和Agent管理器间的"粘合剂"的角色，其主要功能是生成和启动Agent管理器，将构件功能体中组调用的C语言类型的"IN"和"INOUT"参数转换为Java语言类型的参数传递给Agent管理器，将Agent管理器中得到的"OUT"和"INOUT"结果返回给构件功能体，它的结构与组调用表的结构类型无关。相反，Agent管理器的程序结构则完全依赖于组调用表的结构，它是一种特殊类型的移动Agent，因为它固定在客户端，专司管理其它Agent之责，如生成Agent、进行参数传递、在SLOOP和PLOOP结构中判断循环是否继续及收集执行结果。Agent负责迁移到被调构件所在的结点，代表调用构件与被调构件作进程间局部交互，请求其服务，并携带中间状态及最终结果在网络上迁移。组调用表和定位表的输入及输出情况及相互间的关系可用图5表示。

在客户端，当构件体遇到组调用时，会由操作系统将Marshaller动态地装入内存，此后，构件体的执行被阻塞，等待组调用的返回。Marshaller此时获得控制，它会启动Java虚拟机，生成Agent管理器的对象，将参数恰当地传递给Agent管理器的实例变量，并调用执行Agent管理器的startSyn()方法，此后Marshaller等待Agent管理器的run()方法的返回。Agent管理器会根据组调用表的类型，生成Agent并与之交互，其具体的交互过程遵从相应组调用表的语义，当run()方法返回时，Marshaller会取回输出参数的值，并将之由Java转换为C的数据类型，并将控制回给构件体。

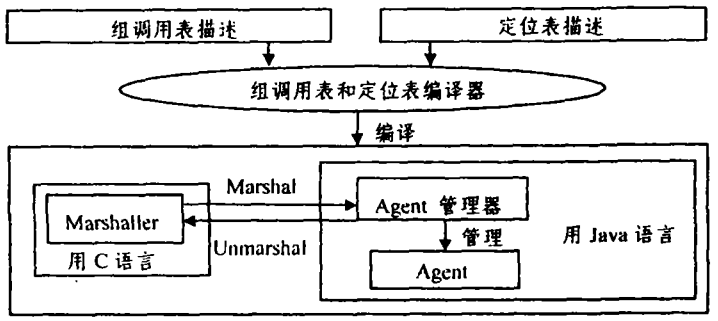

图 5　组调用表和定位表的输入和输出

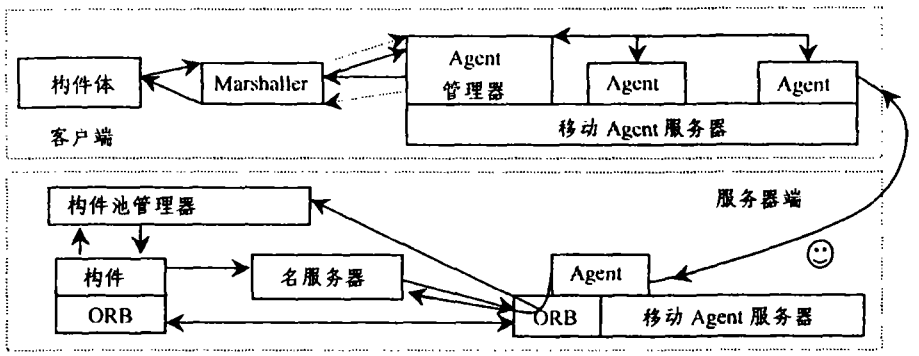

图 6　Cogent 框架中各部分间的交互关系

在服务器端，构件服务器首先向构件池管理器和名服务器注册。当 Agent 到达时，它首先向名服务器查询，将它所携带的名字转换成对象引用，并使用该引用通过 CORBA 的 ORB 机制与构件池管理器交互。如果构件服务器尚未启动，由构件池管理器将相应的构件服务器启动，并将 Agent 的请求传递给服务器。此后，Agent 与构件的服务器通过 ORB 和 IIOP 通信，当在一个站点上的交互完成后，Agent 会重复上述过程，根据其旅行计划迁移到另一个站点或返回初始结点以将结果返回。Cogent 中构件不能直接在操作系统上运行，还需要一些工具的支撑以发挥整个系统的功能。其中，构件池管理器、名服务器和移动 Agent 服务器是必需的，而移动 Agent 监视器和构件发现器是可选的。移动 Agent 监视器负责监视派发的 Agent 的行踪及其生成和消亡状况。构件发现器帮助开发者定位所需的构件，并可自动在运行中填写定位表中的信息，这样可实现动态的后组装。下面分别讨论构件池管理器、名服务器和移动 Agent 服务器的功能结构。

3.2　构件池管理器

构件池管理器负责管理一台主机上的所有构件，在概念上，它类似于 CORBA 的实现池或 COM 中的系统注册表。构件的所有相关信息要在构件池管理器中事先注册方能完全向客户展现其功能，注册过程可采取两种方式：一是为构件开发员提供一系列的 API，他们可以在构件的初始化部分使用这些 API 来向构件池管理器注册该构件，另一种方法是提供工具给开发者，它接受构件的信息为输入，通过与构件池管理器交互来注册。第一种方法属自动注册，它使得构件的管理者免于手工地登录信息，从而方便了构件的配置。构件池的管理器的主要职责是监视来自客户端的请求，如果被请求的构件已经注册并且其服务器不在运行，则构件池管理器自动地、对客户透明地启动服务器并将请求传递给服务器去处理。

3.3　名服务器

名服务器将名字在运行中翻译为相应的引用，它维护了一张构件实例名与其引用间的映射列表。名服务器的必要性出于两个原因：首先，名字易于开发者记忆和使用，而构件引用是一串八位的比特流，其中隐含了位置信息，更适于软件程序去处理。另一个原因是为了达到位置透明性。客户仅持有构件实例的名字，构件实例可自由地移动，只要名服务器能够将名字正确地解析到构件的引用，客户就可不作任何改变而与服务器正常地交互。

3.4　移动 Agent 服务器

移动 Agent 服务器为 Agent 和 Agent 管理器提供了运行支撑平台。它的功能可分为三部分：负责 Agent 的迁移、负责调用和解释执行 Agent 携带的代码，及负责管理 Agent 相关的信息。

SEQ/ALL	$Host_1$	$Entry_1()$
	$Host_n$	$Entry_n()$

图 7　移动 Agent 旅行计划的结构

移动 Agent 服务器目前支持两种 Agent 迁移模式：SEQ 模式和 ALL 模式。SEQ 和 SLOOP 结构的组调用表被翻译成具有 SEQ 模式旅行计划的移动 Agent，PAR 和 PLOOP 结构的组调用表被翻译成具有 ALL 模式旅行计划的移动 Agent，Agent 管理器总是具有 SEQ 模式，因为它不需要移动。在 SEQ 模式中，移动 Agent 服务器将 Agent 迁移到 $Host_1$，调用执行入口方法 $Entry_1()$，然后重复上一过程，直至 $Host_n$ 被访问且 $Entry_n()$ 被调用，这一过程是顺序的，每个结点被依次地访问，各方法被逐个地调用。ALL 模式与之不同，它反映了执行的并行性，移动 Agent 服务器会同时生成 n 个 Agent，$Agent_k$ 会被派发到 $Host_k(k\in[1,n])$，执行方法 $Entry_k()$，这 n 个 Agent 会并发地执行。

Agent 服务器还提供了调用和解释执行 Agent 携带的入

口方法代码的功能，这些入口方法构成了 Agent 的功能主体，而 Agent 的旅行计划作为其迁移向导。此外，Agent 服务器还支持 Agent 信息的管理，如它维护了一个站点上的所有 Agent 的名与引用间的映射，这样 Agent 可以通过名来取得它所感兴趣的任何一个 Agent 的引用。

3.5 Cogent 的四种连线方式

我们设计了四种类型的组调用表：顺序结构、顺序循环结构、并行结构和并行循环结构，以支持四种不同的连线方式。

下面我们详细介绍这四种结构的组调用表：

1）*顺序结构* 图 8.1 表示了组调用表的顺序结构，顺序结构提供了最基本的控制流程。方法按顺序一个接一个地调用，具体说来移动 Agent 一开始根据定位表查出服务接口 I_1 所在的位置，然后移动到该位置，执行方法调用 I1.M1(…)，M1 方法可能是同步的，于是移动 Agent 等待 M1 的完成；M1 也可能是异步的，它立即返回。不管出现哪种情况，Agent 执行后处理 P1（如果 P1 存在）。接下来，移动 Agent 移动到 I2.M2() 所对应的位置，使得能本地调用 I2.M2()。重复上述过程，直到组调用表中最后一个方法被调用，然后返回最初的位置，提交结果给构件体。

SEQ	R1	I1.M1(…)	P1
	R2	I2.M2(…)	P2
	…	…	…

图 8.1 sequential structure

SEQLOOP	Condition	R1	I1.M1(…)	P1
		R2	I2.M2(…)	P2
		…	…	…

图 8.2 sequential loop structure

2）*顺序循环结构* 图 8.2 表示了组调用表的顺序循环结构。顺序循环结构模仿传统程序设计中的循环控制。移动 Agent 在循环中的执行过程与上一部分讨论的顺序结构几乎完全一样。但是当 Agent 完成最后一个方法调用后返回原来的位置，不是简单地提交结果给构件体，而是基于返回的结果计算 Condition 的值，如果值为 true，则再一次执行循环中的操作，直到计算结果为 false 为止，结果才被提交给构件体。

3）*并行结构* 图 8.3 表示了组调用表的并行结构。并行结构很重要，因为它为开发者提供了描述并行处理的能力，这对于提高执行效率尤为重要。当构件体中调用一个并行结构时，控制转换的顺序与上述两部分所说的完全不同。它所产生的 Agent 有 N 个，N 是组调用表中 $I_k.m_k()$ 的个数，每一个 Agent 移动到提供服务接口 I_k 的位置，执行方法调用 $I_k.M_k()$，根据该方法是同步还是异步，决定是等待或是立即返回，并将结果返回到最初的位置。当 N 个 Agent 都返回时，执行后处理 P，执行后才将结果返回给构件体。存在并行时，除非 N 个 Agent 不涉及到共享变量，否则就需要同步。为了组调用表的设计简单，我们规定 N 个 Agent 中无共享变量。

PAR	R_1	$I_1.M_1(\cdots)$	P
	R_2	$I_2.M_2(\cdots)$	
	…	…	

图 8.3 parallel structure

PARLOOP	Condition	R_1	$I_1.M_1(\cdots)$	P
		R_2	$I_2.M_2(\cdots)$	
		…	…	

图 8.4 parallel loop structure

4）*并行循环结构* 图 8.4 表示了组调用表的并行循环结构。并行循环结构与并行结构相比，在所有 Agent 都返回时要计算 Condition 的值，当 Condition 值为 true 时，N 个 Agent 必须再各自移动到目标位置，执行过程调用直到 Condition 的计算结果为 false，才把结果提交给构件体。

组调用表既支持顺序结构（SEQ），也支持并行结构（PAR）。这两种结构的重复就提供了在原有语义基础上描述循环的能力，所以四种结构类型应该可以满足通常的装配要求。另外，为了保证移动 Agent 足够小，上面的组调用表结构不允许嵌套。

4. 与传统构件的比较

由于移动 Agent 具有自治性、移动性、安全性和适应性等特点，在使用移动 Agent 代替传统的连线机制后，能够克服传统连线机制的种种不足之处。我们对目前较为流行的中间件技术和 Cogent 框架进行比较（表 1）。

表 1 各种连线机制的比较

	代码移动	断开式交互	对语言的支持	交互过程	网络负载	效率
JavaRMI	不支持	不支持	支持 Java 语言	多次远程交互	较高	较低
DCOM	不支持	不支持	支持异种语言	多次远程交互	较高	较低
CORBA	不支持	不支持	支持异种语言	多次远程交互	较高	较低
Cogent 框架	支持	支持	支持异种语言	通过 Agent 移动将远程交互变为局部调用	较低	较高

从上表可以看出，Cogent 的连线机制较之传统构件的连线机制有许多优势：首先，由于 Agent 携带代码，这就实现了代码在网络上的迁移，从而小段代码可以传输到大的数据源去处理，降低了网络负载。其次，由于仅在 Agent 迁移时需要网络沟通，这就大大降低了对网络的依赖，提高了应用的健壮性和容错能力。最后，由于起初的多次构件间的远程交互被 Agent 与构件间的局部通信所替代，从而减短了网络延迟，提高了应用效率。

结束语 在 Cogent 系统中，我们采用移动 Agent 作为新的连线机制，具有灵活性和移动性，从而克服传统总线的一些不足。目前我们已经用 Cogent 系统开发了网上投票系统、数

（下转第 56 页）

```
                                    任务集的算法 * /
        IF (success = SUCCESS)THEN Upper = k;
        ELSE Lower = k;
        k=⌊(Lower + Upper)/ 2⌋;
    END WHILE
    RETURN (k + 1;Ω);
END
```

找寻 MNP 仿真实验工作量参数是:失败率是1×10^{-6}/hour;CW=[0.5,1.5];CV=[1,10];ET=[1,200];δ=[1,50]。图2显示:随着任务数目的增加,大多数算法的最小处理器数目也随之增加,而采用图状 DAG 的 RCD 则始终保持3这个值。

4.4 RCD 算法中执行时间和可靠性代价之间的关系

下述试验分析 RCD 的执行时间和可靠性代价之间的关系。ET 有两个范围:[1,100]和[1,200]。从图3可以得出结论:当每个向量 C_i 中的执行时间增加时,系统的可靠性代价也随之增加。

结论 在实时任务调度研究领域中,现有的大部分工作要么就是没有考虑容错性和可靠性问题,要么就是仅考虑系统的异构性,或者假定任务之间是独立的。本文设计了一种可靠性代价驱动的算法 RCD,它能在异构实时系统中进行具有优先权限制的实时任务调度。该算法将可靠性代价用作实时任务调度的一个目标函数,并由可靠性驱动整个实时任务调度过程。同时列举了没有考虑可靠性代价的 AEAP 和 ALAP 算法作为对比。仿真试验的结果表明,依据可靠性代价和作为调度性量度的最小处理器数目,RCD 远远优于 AEAP 和 ALAP 算法。而且该结果还表明较高的异构程度有利于提高 RCD 算法的调度性,而对于 AEAP 和 ALAP 算法则相反。该仿真试验是基于两种类型的实时任务图(DAG),即二叉树任务图和网状任务图,它们是很有代表性的实时任务图。工作量参数要么是基于文献,要么是选择那些能合理代表实际工作量的数值,并能为算法提供一些强化测试。

参 考 文 献

1 Berman P, DasGupta B. Improvements in Throughput Maximization for Real-Time Scheduling, DIMACS: [Technical Report, TR-99-52]. 1999

2 Chang G L, Joosun H, Yang M S, et al. Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling. IEEE Trans. on Computers, 1998, 47(6):700～713

3 Chen Z, et al. An Architecture to Support Collaborative Distributed Heterogeneous Computing Applications: [Technical Report, TR97-26]. Department of Computer Science, University of Bristol, UK, 1997

4 Liu H, Zarki M E. Adaptive Source Rate Control for Real-Time Wireless Video Transmission. Mobile Networks and Application, 1998, 3:49～60

5 Manimaran G, Murthy C S R. An Efficient Dynamic Scheduling Algorithms for Multiprocessor Real-Time Systems. IEEE Trans. On Parallel and Distributed Systems, 1998, 9(3):312～319

6 Shatz S M, Wang J P, Goto M. Task Allocation for Maximizing Reliability of Distributed Computer Systems. IEEE Trans. Computers, 1992, 41(9):1156～1168

7 Son S H, Chaney C. Supporting the Requirements for Multilevel Secure and Real-time Databases in Distributed Environments. Database Security: Status and Prospects, Chapman and Hall Publishing, 1998. 73～91

8 Srinivasan S, Jha N K. Safty and Reliability Driven Task Allocation in Distributed Systems. IEEE Trans. Parallel and Distributed Systems, 1999, 10(3):238～251

9 Chetto H, Silly M, Bouchentouf T. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints. J. Real-Time Systems, 1990, 2(3):181～194

10 Ramamritham K. Allocation and Scheduling of Complex Periodic Tasks. In: Proc. Int'l Conf. Distributed Computing Systems, 1990. 108～115

11 Natale M D, Stankovic J A. Dynamic End-to-End Guarantees in Distributed Real-Time Systems. In: Proc. Real-Time Systems Symp., 1994. 216～227

12 Abdelzaher T F, Shin K G. Combined Task and Message Scheduling in Distributed Real-Time Systems. IEEE Transactions on Parallel and Distributed Systems, 1999, 10(11)

13 Eshaghian M M, Wu Y C. Mapping heterogeneous task graphs onto heterogeneous system graphs. In: IEEE Proc. 6th Heterogeneous Computing Workshop (HCW '97), Geneva, SWITZERLAND April 1, 1997. [14] Sirbu, M. G.; Marinescu, D. C, A scheduling expert advisor for heterogeneous environments. In: IEEE Proc. 6th Heterogeneous Computing Workshop (HCW '97), Geneva, SWITZERLAND April 1, 1997

14 Qin Xiao, Han Z F, Jin H, Pang L P, Li SL. Real-time Fault-tolerant Scheduling in Heterogeneous Distributed Systems. In: Proc. of the 2000 Intl. Workshop on Cluster Computing- Technologies, Environments, and Applications (CCTEA' 2000), Las Vegas, Nevada, USA, June, 2000

15 Maheswaran M, Siegel H J. A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems. In: IEEE Proceedings of the Seventh Heterogeneous Computing Workshop, Orlando, Florida, 1998

(上接第41页)

据检索、并行计算等实例。结果表明,Cogent 系统提供了一种新的基于构件的应用软件系统开发方法。

参 考 文 献

1 Tanenbaum A S. Computer Networks. Prentice Hall, 1989

2 Stevens R. Unix Network Programming, Second Edition, Volume 1. Networking APIs: Sockets and XTI. Prentice Hall, 1998

3 Armstrong J, et al. Concurrent Programming in Erlang. Prentice Hall, 1999

4 Burghart T. Distributed Computing Overview. QUOIN, Cambridge, Massachusetts, 1998

5 Object Management Group. The Common Object Request Broker: Architecture and Specification, revision 2.2, 1998

6 Microsoft. DCOM Technical Overview. white paper, November 1997. http://www.microsoft.com/ntserver/library/dcomtec.exe

7 Microsoft. DCOM Architecture. white paper, July, 1997. http://www.microsoft.com/com/wpaper/default.asp

8 Sunsoft. Java RMI Tutorial, Revision 1.3. JDK 1.1 FCS, February 10, 1997

9 Lu Jian. Some Research on Componentware Frameworks based on Mobile Agent Technology: [technical report of State Key Laboratory for Novel Software Technology]. Nanjing Univ., Nanjing, P. R. China, 1999