Client/Server 计算的协作模型及其开发工具

黄海宁　孙伟伟　夏宽理
（复旦大学计算机科学系　上海 200433）

摘　要　基本的 Client/Server 模型的核心是 Client 向 Server 请求单个独立的服务，为处理 Client 请求复杂服务的情况，本文提出了一个 Client/Server 计算的协作模型，可支持 Client 激活多个子服务以及多个子服务之间的协作交互，并在此基础上设计了一个基于代理的 Client/Server 开发工具。

关键词　Client/Server 计算　协作模型　代理

COORDINATION MODEL & DEVELOPMENT TOOLS FOR Client / Server COMPUTING

Huang Haining　Sun Weiwei　Xia Kuanli
(*Department of Computer Science, Fudan University, Shanghai 200433*)

Abstract　The core of basic Client/Server model is sending a single service request to the server from client. In order to deal with Client requesting complex services, this paper puts forward a coordination model about Client/Server computing. This model supports the activation of several subservices and the cooperation between the subservices. Based on the model, the author designed an agentbased develop tools for Client/Server model.

Keywords　Client/Server computing　Coordination model　Agent

1　引　　言

分布式计算系统是由多个分散的处理资源组成的计算系统，这些分散的资源最小依赖于集中的程序、数据或硬件，在整个系统的控制下协作运行。分布式计算系统在当前软件工业生产中得到广泛的应用，分布式结构使得用户能共享网络中的程序和数据资源，增强了可用性、可靠性，并通过程序或数据的复本技术、并行计算等提高了计算性能，获得这些好处而付出的代价是导致了设计和控制的复杂性。

当前，分布式计算的一个得到广泛应用的重要实例是 Client/Server 计算。Client/Server 计算

黄海宁，硕士生，主要研究领域：软件工程。

首先是一个逻辑概念,客户部分和服务器部分可存在或不在同一物理机器上,更精确地说,Client/Server是同时执行的程序之间相互作用的一种特殊形式,具体表现为客户向服务器发出请求,服务器执行请求并返回相应的结果至客户。然而,Client/Server并不是分布式计算所追求的最佳目标,基本的Client/Server模型的一个重要限制是:它强调的是客户向服务器请求单个独立的服务,因此限制了Client/Server的分布能力。对于在客户同时激活多个子服务,要求多个服务器协作地完成客户所要求的任务的情况下,基本的Client/Server模型无法进一步描述。本文对基本的Client/Server模型进行了扩展,提出了一个Client/Server计算的协作模型,并进一步给出了支持该模型的软件开发工具。

2 Client/Server协作模型

2.1 模型概述

所谓协作,在不同的上下文环境下有不同的意义。广义地说,是将各项相互关联的子任务集成地加以控制,以完成更广泛和复杂的任务。具体地,它包括如何将任务分解成独立的或相关联的子任务,如何将子任务分配给各有关过程去执行,同时必须对协作执行进行控制,它还包括采用何种资源分布策略以及如何实现资源的共享等方面。

在基本的Client/Server模型下,如果一个客户使用多个服务器,他必须负责按自己的需求去使用它们,客户必须预先知道运行所需服务的服务器的地址,同时服务器进程的协作也必须由客户机自己来控制(如图1)。对于这种模型,当服务器增加或减少等情况而引起资源分布改变时,必须更改包括客户和服务器的几乎所有的程序,这种存取问题在异种系统中变得特别突出,因为请求服务的客户可能与不同的服务器和平台打交道。

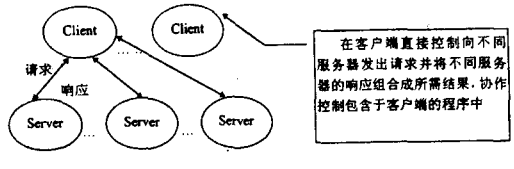

图1 基本Client/Server模型对多服务器的请求

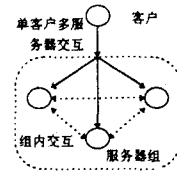

图2 单客户多服务器组交互模型

为了保证灵活性和可用性,一个好的分布式系统,其分布式操作对用户应当是完全透明的,在Client/Server模型下,服务器的透明性是指服务器地址等细节对于客户的不可见性,让客户自己去控制服务器的协作执行是不透明的。

对于在一个客户与多服务器协作交互的情况下,我们使用服务器组来表示一组服务器,同组的服务器用于协作地完成一个功能集合。这里的服务器是一个逻辑概念,并不是物理上的服务器,单个服务器可以根据不同的运行状态可以属于几个不同的服务器组。使用服务器组的概念可以刻画单客户多服务器交互的模型(如图2),将不同服务之间的交互概括为服务器组内的交互,我们称之为组内交互。

客户与服务器之间的交互依赖于请求和响应的网络通讯模型,可能是阻塞的或非阻塞的方式。阻塞方式有利于客户进程和服务器进程的同步,而非阻塞通讯具有并行性,可实现客户进程和服务器进程的异步通讯。在Client/Server协作模型中,发出同步请求的客户将阻塞,直到接收到服务器的响应。阻塞方式的实现比较简单,但是对于客户进程的处理能力来说是低

效率的。非阻塞通讯方式使得客户在发出请求后能执行其他操作,从而提高了执行效率,但是也增加了为获得服务器响应的控制复杂性。

为了解决 Client/Server 计算的协作问题,首先必须解决资源分布问题。资源包括数据资源和程序资源。我们引入对象的概念来描述分布于各个节点的数据资源,一个对象由特定的属性以及可以发生在该对象上的操作所组成。我们使用对象组来描述具有共同的状态特征的一组对象,运行在一个对象组上的所有操作进程构成了一个进程组,对该组对象进行操作的进程都属于该进程组,即所有引起该对象组状态的操作进程构成一个进程组。当进程组中的进程用于向客户提供服务时,我们称该进程为服务进程,进程组中的所有服务进程构成了服务器组。

2.2 协作管理器

在决定了资源分布策略以后,必须建立一个管理客户请求并响应请求的机制。我们定义一个协作管理器来接受所有来自于客户端的请求,由该协作管理器控制在不同的服务器上协作地执行请求,并将不同的服务器上获得的运行结果加以组合,然后由协作管理器将最终结果传送至客户端。

使用协作管理器的好处是:

(1) 客户的代码独立于服务器组中服务器的数目,也即独立于 Client 与 Server 的布局结构,使得客户代码仅关心与最终用户的界面操作;

(2) 服务器的代码独立于同服务器组中其他服务器的数目,也即使得服务器代码最大限度地独立于数据资源的分布及服务器组的划分,而仅仅关心其所提供的服务;

(3) 用于提高性能和容错能力的代码被封装至协作管理器上,协作算法不属于客户和服务器代码的范围,因此不必改动客户和服务器的代码就可以提高服务的性能和容错能力。

(4) 同时提供一个基本的 API 界面供应用程序调用,简化客户程序和服务程序的编程。

2.2.1 请求代办者

协作管理器首先必须管理来自于客户的请求,我们定义一个请求代办者,它负责协调客户应用程序需要的服务和服务器应用程序所能提供的服务。请求代办者使客户不必关心在哪里和如何获得特定服务。所有分布于系统各节点的应用程序通过"请求代办者"注册它所能提供的服务和需要请求的客户接口,注册服务时必须提供服务的地址及接口。通常,请求代办者管理一个名字服务或目录。基于"请求代办者"模型,服务可以在运行时动态注册或在编译时静态注册。请求代办者可以使用如下三种设计模型之一完成 Client/Server 交互。

完全代办模型:客户传递请求至代办者,代办者将客户请求传递至相关的服务器程序,并取回响应,最后将响应传递至请求客户。

句柄代办模型:客户传递请求至代办者,代办者处理该请求并返回一个句柄至客户,该句柄包含了与服务器交互的关于该服务的所有信息(包括名字、网络节点地址、接口等),客户通过该句柄与服务器应用程序交互。

混合代办模型:同时支持以上两种模型,让客户在发出初始请求时用参数来选择。

完全代办模型提供了控制所有客户请求和服务器响应的场所,因此可以检测失败的交互并可重试,容错性和可恢复性不必由客户实现。缺点是消息传递的集中性有可能使性能降低。句柄代办模型减少了潜在的瓶颈问题,来自于服务器的响应避开了代办者而可以直接传递至客户。而混合代办模型则同时提供了两种机制供客户灵活地选择使用。

2.2.2 服务控制器

协作管理器中的核心部分是服务控制器,客户发出一个隐含服务请求,这个隐含服务请求不能由一个服务器直接完成,服务控制器必须将客户请求映射或分解成各个子服务请求并分析存在于各子服务之间的依赖关系,然后协调各个相关服务器的活动并收集响应,最后将这些响应组合成一个单一和一致的响应并传递给客户。

如果各子服务相互独立,则可采用非阻塞模型,向有关的服务器发出请求并行执行,最后合并结果。相反,对于相互依赖的服务,必须具有同步约束的机制,可采用阻塞的通讯模型。

下面列出了服务控制器的一个算法。

(1) 映射或分解一个客户的隐含服务请求至子服务请求。

(2) 为子服务请求识别服务器。

(3) 为子服务之间的每个依赖约束建立一条记录。

(4) 循环…直到所有子服务完成

(A)对于与当前子服务无依赖约束的所有独立子服务,并向相应服务器发出子服务请求;

(B)从那些子服务所在的服务器收集响应;

(C)更新依赖约束的记录。

(5) 组合所有响应至客户所需的结果。

(6) 将最终结果传递至请求客户。

算法中(2)、(4A)、(6)调用了请求代办者的功能。

为了描述隐含服务的依赖关系,我们设计了一个隐含服务描述语言,服务控制器使用上述算法解释执行隐含服务描述,并记录执行的中间状态和响应,直到所有子服务执行完毕并返回结果。该描述语言需要定义的是:

(1) 隐含服务的各个子服务;

(2) 定义和设置中间变量(用于获取子服务请求的调用参数和子服务结果);

(3) 定义子服务协作执行的控制结构(依赖关系);

(4) 激活其他描述。

我们用一个例子来说明隐含服务描述语言。

```
# 隐含服务描述语言的一个例子
# 读入服务参数
set p1 = argv[0]
set p2 = argv[1]
set p3 = argv[2]

# 调用子服务 SR0 并设置中间变量 a1
set a1 = SR0(p1)

# 分支结构
if a1{
  set a2 = SR1(p2,"const para")
} else{
  set a2 = SR2(p3,a1)
```

```
}
# 调用另一个描述并返回值
set a3 = interpret aScript
parallel{
set a4 = SR3(a3)
set a5 = SR4
}
# 组合结果并返回
return a2 + a5 + a4
```

2.2.3 服务器组管理器

我们定义了一个服务器组管理器作为协作管理器的一部分用于维护服务器组的静态和动态信息。

在分布式系统中,资源分布于各节点的计算机上,有两种管理资源的方式,一种是采用一个节点统一管理,称为集中分布管理方式,另一种是由多个节点共同管理,称为全分布管理方式。集中分布管理实现比较简单,而全分布管理的实现比较复杂,一般采用混合管理方式,即对一些资源采用全分布管理,而对另外一些资源实行集中分布式管理。

我们在对分布数据资源进行对象分析的基础上,建立资源分布策略,并为各个资源对象建立操作进程组并提炼出服务进程构成服务器组。我们使用服务器组管理器来管理服务器组,由服务器组来管理分布的数据资源,既具有全分布管理的特点,又保留了集中管理的简洁性。

另外,考虑到服务器组的动态性,服务器组管理器还必须提供动态创建、更改、取消、合并一个服务器组的功能。

2.3 服务器组协作计算开发工具

实现分布式 Client/Server 模型需要诸如远过程调用(RPC)和消息传递等工具以建立和控制网络上应用程序间的通讯,消息系统通常提供了一个发送消息至发送队列和从接收队列中接收响应的应用程序接口(API)。这种设计增加了应用程序的额外控制能力,导致了应用程序独立性降低,也影响了分布式系统的模块化、可维护性和可扩展性。为解决这个问题,通常在应用程序与网络 API 之间提供一个代理(Agent),这些代理提供了清楚的、结构化的构件以调度应用程序、通讯核心,以及执行应用集成或分布控制的其他任务,即代理提供了一个类似 RPC Stub 程序的中间角色的功能。代理可以用于实现基本的 Client/Server 模型(如图 3)。

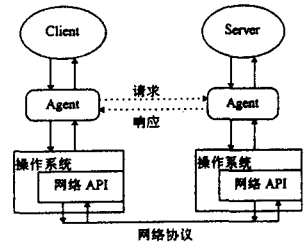

图 3 基于代理的基本 Client/Server 模型的实现

基于 2.2 节所描述的协作模型,我们开发了一个协作计算开发工具,其核心包括了基于代理机制的协作管理器和协作模型应用程序接口。

协作管理器是一个逻辑概念,在实现方式上,既可以建立一个协作服务器节点,也可以分布于多个节点上。但是,如果在一个节点上实现,协作管理器容易成为系统的瓶颈,在分布式系统中,我们要尽量将协作管理器分布在多个节点上,但这种分布性势必增加了控制复杂性。

我们将协作管理器建立在服务器组的基础上,即在系统运行的某一个状态,每个服务器组都对应一个协作管理器来管理和协作该服务器组上服务的运行。同时,一个协作管理器可以

管理一个或多个服务器组,协作管理器可以在或不在同一个服务器组所在的节点上(如图4)。

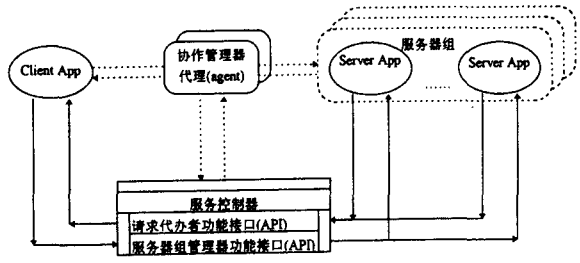

图4 协作模型实现

基于上述观点,我们开发了一个协作计算开发工具(FD-CDT),它包括了一个基于代理机制的协作管理器和协作模型应用程序接口。

表1列出了请求代办者API的基本接口;

表2列出了服务器组管理器API的基本接口。

表1 请求代办者API主要函数

函 数	功 能
基本函数	
RBRegisterApp	注册一个应用程序
RBRmvApp	注销一个应用程序
RBRegisterService	注册应用程序所能提供的服务(服务名,服务地址,参数)
RBRmvService	注销某服务的注册(服务名)
RBRegisterRequest	注册应用程序需要的隐含服务请求(服务名,隐含服务描述)
RBRmvRequest	注销应用程序的隐含服务请求(服务名)
为客户提供的函数	
RBInvokeService	采用完全代办模式请求服务(服务名,参数),返回结果
RBGetServiceHandle	采用句柄代办模式为请求服务作准备(服务名),返回服务句柄
RBRequestUseHandle	采用句柄代办模式请求服务(参数)
为服务器提供的函数	
RBReply	采用完全代办模式返回结果(服务名,结果)
RBReplyUseHandle	采用句柄代办模式返回结果(服务名,结果)

表2 服务器组API主要函数

函 数	功 能
SGCreate	创建一个服务器组(组名,服务名表)
SGDelFrom	从一个服务器组删除一个服务(组名,服务名)
SGAddTo	向一个服务器组插入一个服务(组名,服务名)
SGMerge	合并多个服务器组(组名表)
SGRmvGroup	撤销一个服务器组(组名)

(下转第59页)

3.5 其 他

除了以上的例子之外，还有的研究者在网络路由算法中采用 Agent 方法；采用协作的方式进行 ATM 流量控制；通信系统的操作支持系统等。

4 结 论

随着研究的深入，智能 Agent 领域的研究必将不断扩展和深入。采用精神状态来进行复杂系统的抽象为人工智能提供了新的思路，并提供了跨平台互操作的可能性。Agent 的协作和协调是多 Agent 系统性能超过单一智能系统的关键，仍将是研究的主要对象。在 Agent 系统的实现上，主要考虑多 Agent 的组织结构及 Agent 体系结构，Agent 设计语言，相关的领域包括面向对象的方法和分布式对象。

计算机网络和通信系统具有分布性的特征，是智能 Agent 方法的典型应用领域。主要将用于网络的优化、参数采集、网络管理等方面。

参 考 文 献

[1] M. Wooldridge and N. R. Jennings, Intelligent Agents: Theory and Practice.

[2] H. Velthuijsen, Intelligent Agents for Future Telecommunications Systems.

[3] H. Velthuijsen, Distributed AI and Cooperative Systems for Telecommunications in: J. Liebowitz and D. Prereau(eds.), Worldwide Intelligent Systems: AI Approaches to Telecommunications and Network Management, pp. 227 ~ 260, 1994, IOS Press, Amsterdam.

[4] H. S. Nwana and M. Wooldridge, Software Agent Techologies, BT Technol. J., Vol. 14, No. 4, Oct., 1996.

[5] D. G. Griffiths and C. Whitney, The Role of Intelligent Software Agent in Integrated Communications Management, BT Technol. J., Vol. 9, No. 3, Jul., 1991.

[6] The DARPA Knowledge Sharing Intiative External Interfaces Working Group. Specification of the KQML Agent - Communication Language.

(上接第 6 页)

3 结 语

Client/Server 是当前软件开发中的重要领域，我们针对 Client 复杂的服务请求，提出了协作计算模型，并在此基础上实现了协作模型开发工具 FD - CDT，正如文中所讲的，我们在 Windows95 环境下用 VisualC + + 实现了 FD - CDT 的一个原型。为简化服务描述，我们在文中仅给出了基本的服务接口，为进一步实用化，我们正研究将该工具与多种数据库管理系统的结合并进一步完善 Client/Server 开发的系列工具。

参 考 文 献

[1] G. Coulouris and J. Dollimore, Distributed Systems: Concepts and Design, Addison - Wesley, Reading, Mass., 1988.

[2] H. Bal, J. Steiner and A. Tanenbaum, "Programming Languages for Distributed Computing Systems", ACM Computing Surveys, Vol. 21, No. 3, Sept., 1989, pp. 261 ~ 322.

[3] Carriero N. and Gelernter D., "Coordination Languages and Their Significance", Comm. ACM Vol. 35, No. 2, Feb., 1992, pp. 97 ~ 107.