基于OGSA网格服务的工作流调度代理的设计*)

李 锐 申德荣 于 戈

(东北大学 信息科学与工程学院 沈阳110004)

摘 要 网格服务是近期对Web服务技术的加强和扩展。它为Web服务提供了服务发现、生命周期、状态管理、创建与销毁、事件通知以及引用管理等功能。本文详细地介绍了一个基于网格服务的工作流调度代理的设计与实现。它充分地利用了网格服务的新增的特性,改进了传统的面向服务的调度代理实现机制,并提供了动态创建、销毁工作流实例,挂起、恢复和终止流程等新功能。

关键词 网格服务,OGSA,调用代理,工作流

Design of Workflow Scheduling Broker Based on OGSA Grid Service

LI Rui SHEN De-Rong YU Ge

(School of Inormation Science & Engineering, Northeastern University, Shenyang 110004)

Abstract Recently, Grid Service is the enhancement and extension of Web Services technology. It provides many new functions to Web Services, such as service discovery, lifecycle management, state management, service creation and destruction, event notification and reference management and so on. In this paper, the design and implementation of workflow scheduling broker based on Grid Service are introduced. It improves the implementation mechanism of the traditional service-based scheduling broker by using the new features provided by Grid services, and provides a number of new functions such as dynamic creation and destruction of workflow instance, suspension, resumption and termination of workflow instance.

Keywords Grid service, OGSA, Scheduling broker, Workflow

1 引言

伴随信息技术的发展,产生了越来越多的数字化资源,这些资源通常是分散的、异构的和冗余的。为了充分利用已有的计算资源和信息资源,资源集成技术逐渐成为一个重要的课题。而工作流是处理面向过程的应用集成的主要方式。应用工作流技术进行应用集成时,需要对异构的应用资源进行整合,这就要求描述语言的定义和调度代理的实现脱离资源的实现细节,并且以一致的方式访问异构的资源。Web服务满足了这种需求。Web服务通过WSDL定义封装良好的接口来隔离资源的使用与实现细节,同时提供了与语言和平台无关的访问方式,便于对异构资源进行整合。因此,从Web服务技术面世以来,面向服务的工作流技术迅速发展,出现了BPEL4WS[1]等工作流定义语言和BPWS4J等调度引擎的实现。

然而,由于Web服务的无状态性,使得面向服务的工作流系统存在着一些不足。比如,工作流实例必须通过唯一的数据关联集进行确定,并且在执行期间不能更改;工作流的内部状态管理困难等。网格技术与Web服务相结合弥补了上述不足。全球网格论坛(Global Grid Forum, GGF)提出的开放网格服务体系结构(Open Grid Service Architecture, OGSA)将Web服务技术同网格相结合,对Web服务进行了扩充和加强。OGSA定义了一组标准Web服务端口(portType)和约定,为Web服务提供了服务发现、生命周期、状态管理、创建与销毁、事件通知以及引用管理等功能。应用OGSA技术规范可有效地实现工作流实例的动态管理,为支持Web环境下的分布、动态、自治、异构的资源的集成提供了良好的技术支持[2]。

我们的系统就是建立在OGSA网格服务基础之上的,它充分利用了OGSA为Web服务提供的扩充功能,大大提高了工作流系统中调度代理的动态性与灵活性。系统引入了调度引擎实例服务,管理工作流的整个生命周期,引擎实例可以根据需要灵活创建,并且在实例中维护工作流的状态信息,从而解决了传统的面向服务的工作流系统中的一些不足。

2 分布式代理结构介绍

调度代理是系统中最繁忙的部件,调度代理的性能会直接影响到整个系统的作业吞吐能力和可靠性,保证调度代理的可靠性(availability)和可伸缩性(scalability)是保证系统提供服务的质量重要一

*)课题得到国家863计划CIMS主题(编号:2003AA414210)、国家自然科学基金(编号:60173051)资助。李 锐 硕士研究生,主要从事Web服务、网格服务集成和工作流方面的研究。

环。在大型的工作流系统中，同时有大量的流程在执行，不同时刻系统的负载相差很大。为了保证系统在不同负载下都提供可接受的性能和成本，通常有多台主机组成分布式系统，在不中断系统服务时允许主机加入和退出系统。

分布式代理系统通常有两种组织方式，平面分布式结构和层次分布式结构[3]。

在平面分布式结构中，每台代理主机都处于平等的地位之上，完成相同的功能。主机之间通过通信维持一致的状态，一台主机退出系统时，其他主机可以管该主机的工作。

平面结构的代理系统中有如下关键技术：(1)可以使用 Gossip 算法[2]维护对等主机间的状态一致性。(2)当主机失效时，各分布式主机需要对该事实取得一致性结论，可采用 FloodSet 算法或 EIGStop 算法或它们的优化版本实现[4]。(3)当有主机退出或失效时，需要采用适当的算法来选举接管服务的主机，通常可采用的算法有：同步 LCR 算法和异步 LCR 算法[4]。

平面结构的优点：系统的可靠性高，只要有任何一台主机可以工作，系统就可以维持工作。存在的缺点：状态维护开销过大。为了维护 n 个分布式主机的状态一致性，需要发送的消息数为 $n(n-1)/2$；通信开销的增长速度是平方复杂度，因此系统的可扩展性受到了限制。

在层次分布式结构中，采用多个分布式调度主机，称为局部代理(Local Broker)。调度引擎运行于局部代理之上，负责流程的调度。同时，增加一个集中式的调度节点，它的任务是将用户提交的作业分配到局部代理上执行，监视并保存每个局部代理的运行状态。

层次代理结构的工作方式是：由全局代理接受用户的请求，并根据负载情况将流程分配给有效的局部代理(负载平衡 workload balancing)，由局部代理进行具体的调度工作，局部将代理定期向全局代理发送通告，并将自身正在执行的流程的状态通报给全局代理。在主机退出或失效时，全局代理可以利用保存的流程状态，在其他局部代理主机上恢复作业的执行。

层次代理结构的优点：通信开销较低，故障检测容易，同时由于通信复杂度的增长是线性的，系统的可扩展性更强。存在的不足是：(1)可靠性较低，系统依赖单一的全局代理；(2)单点瓶颈，全局代理是唯一的，随着系统的增长，全局代理将逐渐成为整个系统性能的瓶颈。

3 代理的体系结构

在传统的基于 Web 服务的工作流调度引擎中，工作流是部署在主机上的特定的 Web 服务，使用者通过该服务的特定操作启动工作流，并通过定义好的接口同流程交互。它要求高级用户事先编写对工作流描述(如：.bpws 文件)，其部署在服务器之上，修改流程定义比较困难，难以适应一些用户定制的和临时的需求。同传统流执行引擎不同，本系统中的执行引擎是一个 OGSA 标准的网格服务，它由运局部代理主机上的引擎工厂(Engine Factory)创建。引擎工厂是一个部署在局部代理主机上的 Web 服务，它根据用户的工作流描述，解析出流程的模型，并使用该模型作为创建调度引擎实例。引擎工厂还同时创建一个资源代理对象，用于查找执行中所需的资源服务。调度引擎服务按照给定的流程描述执行，只供本次执行的任务使用。这样，无需部署和代码生成过程，用户可以针对特定的或临时的需求灵活地定制任务。

另外，由于 Web 服务自身没有内部状态，面向服务的工作流通常使用数据关联集(corelationSet)的方式判断所到来的请求对应的工作流实例，这制约了工作流定义的灵活性。而网格服务可以被动态地创建和销毁，并可以在生命周期内维护内部的状态信息。因此，使用网格服务作为调度引擎可以不需要数据关联集，只要用户使用创建时获得的 GSH 进行访问就可以访问特定的流程实例。

局部执行代理的结构如图1所示。

作为调度引擎的网格服务不仅提供了 OGSA 中定义的标准端口(如：GridService 等)，还实现了三个功能端口(portType)，用户和系统通过这三个端口对调度引擎进行访问。下面分别介绍各个端口的定义及功能。

(1)流程执行控制端口(ProcessController)

定义：

```
interface
ProcessController {
    void suspend();
    void resume();
    void terminate();
}
```

功能：

本端口由网格门户主机上的流程存根(Process Stub)访问。通过本端口可以挂起(suspend)、恢复(resume)或终止(terminate)流程的执行。用户可以利用网格门户站点上提供的操作界面来访问该端口，对自己提交的作业的执行进程控制。

(2)流程状态监控端口(ProcessInspector portType)

定义：

```
interface ProcessInspector {
    ProcessStatus getProcessStatus();
    ProcessState getProcessSnapshot();
}
```

功能：

本端口由局部代理主机上的流程状态收集器(Process State Collector)访问。该收集器收集并汇总代理主机上运行的所有流程的状态,以便同其他代理主机之间进行状态同步。

getProcessStatus 方法可以获得流程的运行状态,该状态可能是以下几种值:

PROCESS_STATUS_RUNNING:正在调度或执行定义的活动;

PROCESS_STATUS_RECEIVING:正在等待用户输入数据,并重新激活流程;

PROCESS_STATUS_SUSPENDING:接到suspend 命令,正在等待运行中的活动完成;

PROCESS_STATUS_SUSPENDED:调度引擎挂起;

PROCESS_STATUS_TERMINATING:接到terminate 命令,正等待运行中的活动完成;

PROCESS_STATUS_TERMINATED:调度引擎运行终止;

PROCESS_STATUS_SUCCESS:流程执行的所有操作已成功完成;

PROCESS_STATUS_FAILED:流程执行出错结束。

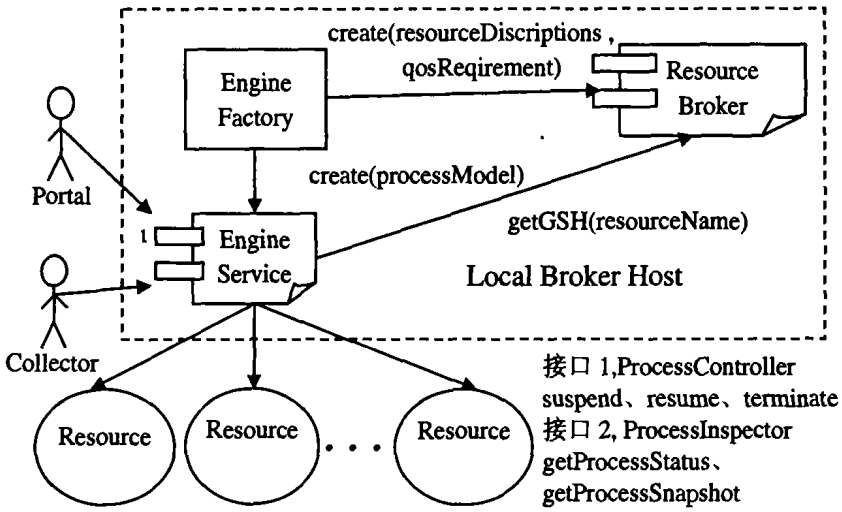

图1 局部代理执行引擎结构

getProcessSnapshot 方法可以获得引擎服务内部状态的快照,包括变量池中变量的值,流程使用的所有操作的 GSH,以及流程中活动(Activity)的执行状态(尚未执行、正在执行、执行完毕)。基于这些状态,我们可以将调度引擎在不同的局部代理主机转移。

(3)用户定义端口(User Defined portType)

用户定义端口是用户访问工作流的接口,用户通过调用该端口中的方法来启动工作流,或为流程继续执行提供必要的输入数据。用户定义端口是在用户提交的流程描述文件中定义的,包含在〈interfaceDefination〉段中。我们采用 WSDL 标准定义流程的用户端口,这样,工作流的用户接口就和普通的 Web 服务的接口定义相一致。

4 执行引擎的设计

在执行引擎服务的运行过程中,工作流的描述模型被转换为一系列相互关联的活动(activity)。执行引擎根据活动的定义,在条件满足时执行相应的动作,并根据执行的结果决定将要执行的活动。执行引擎还包括称为变量池(variable pool)的数据容器,流活动所要执行的操作将从其中获得输入,执行结果可能改变其中变量的值。

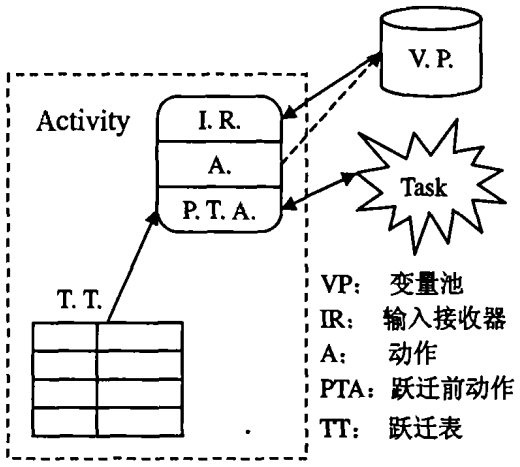

图2 活动结点的结构

活动(activity)由四个部分组成:输入接收器(Input Receiver,I. R.)、动作(Action,A.)、跃迁前动作(Pre-transition Action,P. T. A.)和跃迁表(Transition Table,T. T.),如图2所示。I. R. 判断执行条件是否为真,若为真,从变量池中获取所需的数据,并激活执行;A. 是活动所要执行的任务,比如,调用资源的指定操作,给指定变量赋值等;P. T. A. 将在跃迁前执行,它根据活动的定义执行一些额外

的操作，比如，向特定活动发出消息等；T. T. 中记录着动作执行结果和下一状态之间的对应关系，在活动执行完毕之后将根据跃迁表中记录的信息跃迁到新的状态。

流程中的活动有不同种类的动作，根据种类不同，我们可以将活动分为以下类型[6]：

赋值活动(Assign)：将指定变量或变量的一部分赋给另一个变量；

调用活动(Invoke)：调用指定的资源服务上的特定操作，并获取返回的结果；

接收活动(Receive)：等待用户给出执行所需的数据，并将数据保存在指定变量中；

反馈活动(Reply)：将流程执行的结果(通常是一个变量)反馈给调用者；

等待活动(Wait)：停止流程的执行进程，直到特定条件得到满足；

空活动(Empty)：什么也不做，通常用作占位操作；

抛出错误(Throw)：在流程执行状态异常时产生错误，将错误信息写入指定的变量，执行后流程将跃迁到对应的错误处理活动；

终止活动(Terminate)：立即停止流程的执行过程，释放所占用的所有资源；

复合活动(Compound Activity)：复合活动是一种特殊的活动。复合活动包括一系列活动，它们以特定的方式进行组织。从流程结构上来看，复合活动整体可以被看作是一个简单活动。

复合活动通常是一个控制结构，比如选择或循环，也可能是流程中的一个有事务要求的部分。复合活动执行的动作就是按照结构的定义和前驱活动的结果激活复合活动中包含的相应子活动。复合活动可以嵌套。根据复合活动的组织方式，我们将它活动分为以下类型：

并行处理(Flow)：由两个或两个以上并行执行的活动组成；

原子事务(Atom)：由两个或多个活动组成，这些活动符合 All-or-nothing 原则；

业务事务(Scope)：包含一个或多个普通活动，以及相应的补偿活动，首先执行普通活动，如果执行失败，将执行相应的补偿活动；

顺序处理(Sequence)：由按照顺序执行的一系列活动组成；

选择处理(Switch)：根据活动的输入，在一组活动中选择一个进行执行；

循环处理(While)：在指定条件满足时，反复执行一个活动。

在运行时，整个调度引擎类似一个有限状态自动机，每个活动是自动机中的一个状态，动作是状态对应的操作，状态机将根据模型定义和动作执行结果决定其的下一个状态，即下一个被执行的活动。在解析过程中，根据活动的父活动的定义产生跃迁表，而执行时只需要按照跃迁表行就可以了，这样就简化减轻了调度部分的复杂度，提高了调度引擎的运行效率。

结束语 本文介绍了一个支持 OGSA 网格服务的工作流调度引擎的设计，利用网格服务提供的扩展功能，增强了面向服务的工作流系统的易用性和灵活性。系统引入了虚拟资源的概念，通过对资源的抽象和虚拟化实现对资源的动态集成，充分地利用了 IT 资源，达到了节省成本和提高效率的目的。探讨了分布式调度代理结构，提高了系统的可靠性和可伸缩性，使其可适应大规模、高负载和突发性强的环境。利用自动机原理实现了工作流的调度引擎，使用活动定义工作流中要执行的基本任务，并通过引入复合活动处理工作流中的流程控制和事务结构。

参 考 文 献

1 Leymann F, Roller D, Schmidt M-T. Web services and business process management. http://www.research.ibm.com/journal/sj/412/leymann.pdf, 2001, 3

2 Foster I, Kesselman C, Nick J M, Tuecke S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www-fp.globus.org/research/papers/ogsa.pdf, 2002, 6

3 Subramani V, Kettimuthu R, Srinivasan S, Sadayappan P. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests. http://csdl.computer.org/dl/proceedings/hpdc/2002/1686/00/16860359.pdf, 2002, 7

4 Lynch N A. 分布式算法. 机械工业出版社, 2004, 1