文章编号:1673-9868(2012)11-0100-08

基于分布式多引擎架构的网格工作流管理系统①

赵 钢

西安航空职业技术学院,西安 710089

摘要: 主要研究面向服务的网格工作流管理系统,扩展了服务流程定义语言BPEL,使其支持将物理服务和虚拟服务作为原子服务参与构建工作流服务;提出了分布式多引擎系统架构;基于$M/M/m$排队模型,设计了负载均衡调度算法,能够在各引擎之间进行作业优化调度.实验结果表明,与集中式单引擎系统相比,分布式多引擎系统减少了作业平均响应时间,增强了系统负载能力和可靠性,提高了作业吞吐率和执行成功率,同时具备良好的可扩展性.

关 键 词: 网格工作流;$M/M/m$排队模型;负载均衡;作业调度

中图分类号: TP393.028 **文献标志码:** A

工作流管理系统(workflow management system, WFMS)是指借助计算机系统来定义、创建、执行和管理工作流实例[1].通过解释和执行事先定义的流程逻辑,工作流实例在多个流程参与者之间被调度和执行.网格计算是将地理上分布的、异构的各种资源通过高速网络连接并集成起来,实现计算资源、存储资源、数据资源、信息资源的全面共享.其根本特征是资源的共享和协作,消除资源和信息孤岛.目前的网格系统大都基于面向服务的技术来实现[2],其原因在于遵循标准协议(WSDL,SOAP,UDDI等)的Web服务和有状态的WSRF服务屏蔽了底层异构的物理资源和软件环境,成为能够被动态发现和集成的服务组件实体.因此,遵循服务标准实现的网格服务不仅能够跨语言、跨平台互相访问,还可以根据流程逻辑需要,由独立存在的具有原子事务ACID(Atomic,Consistent,Isolated,Durable)特性的服务(原子服务)扮演一定角色,参与编排生成能处理流程事务的大粒度复合服务,即网格工作流服务.

网格工作流管理系统[3]是指在网格环境下,借助工作流语言,定义工作流服务(复合服务).接收用户对复合服务的调用请求(工作流作业),按照流程逻辑,对参与的原子服务进行调度和执行,最后将作业执行结果返回给用户.系统一般包含一个或多个引擎.工作流引擎是基于某种工作流语言规范实现的中间件,基本功能包括:解释流程定义,创建工作流实例,执行工作流作业,在工作流参与者(原子服务)之间进行交互和调度,提供监控和安全保障机制等.

当前,学术界对网格工作流的研究主要集中在两个方面:一是提出和制定网格工作流定义语言规范;二是开发和实现遵循某种规范的工作流引擎和执行管理系统.

在网格工作流定义语言规范方面,GSFL(Grid Services Flow Language)[4]是由美国Argonne国家实验室提出的一种符合OGSA框架[5],用于描述和定义网格服务工作流的语言规范,其借鉴了工业界对标准Web服务之间交互访问和集成的经验,并将其扩展用于网格服务工作流.AGWL(Abstract Grid Workflow Language)[6]是澳大利亚Aurora网格项目组提出的一个基于XML在抽象层描述和定义网格

① 收稿日期:2011-04-24
基金项目:陕西教育厅基金资助项目(11JK0487).
作者简介:赵 钢(1977-),男,陕西西安人,讲师,硕士,主要从事软件工程教学及研究.

工作流的语言规范，它通过对网格资源抽象描述和定义，提供了一系列抽象逻辑接口，使用户可以对底层异构的网格物理资源和软件环境实现流程透明访问. OWL-WS[7](OWL for Workflow and Services)是由欧盟 NextGRID 项目提出的基于本体论和语义工作流模型的网格工作流规范，支持抽象的和具体的服务工作流模型.

在网格工作流引擎和执行管理系统方面，GridAnt 是 Argonne 实验室开发的一个基于客户端对流程进行管理的工作流系统，其特点是网格用户通过客户端主动管理工作流实例而无需服务提供者参与. GridFlow[8]是一个灵活的、基于 Agent 资源管理方式的网格工作流系统，提供全局网格工作流管理(ARMS)和本地网格子工作流调度，功能包括了工作流创建、仿真、调度、执行、监控、冲突解决等. DAGMan (Directed Acyclic Graph Manager)是 Condor 项目组开发的网格工作流管理系统，其主要用于对有复杂依赖关系的网格作业进行流程管理，通过 DAG 对流程进行建模，不支持循环等流程逻辑.

本文提出并实现了一个面向服务的分布式工作流管理系统——ServiceFlow，能够为各种科研计算和复杂事务流程处理中的工作流作业提供执行环境.

1 ServiceFlow 系统结构

1.1 总体结构

如图 1 所示，ServiceFlow 系统主要由 4 大模块构成，分别是：系统客户端(Grid Workflow Client)、分布式工作流引擎(Distributed Workflow Engines)、服务动态选择代理(Service Selection Agent)、服务容器(Service Container).

客户端是系统对外界提供的访问入口，采用基于 B/S 的 Web 页面架构，包括系统管理员界面，工作流引擎注册界面，复合服务部署界面，复合作业提交界面，引擎、服务和作业的监控界面，状态通知界面，作业结果返回获取界面等.

分布式工作流引擎是整个系统的核心部分，它接收来自系统客户端的各种请求，按照排队模型和作业负载均衡调度算法在多个引擎之间进行作业分派，对复合服务进行多点部署，对引擎、服务和作业实施分级监控、引擎信息注册、日志管理等.

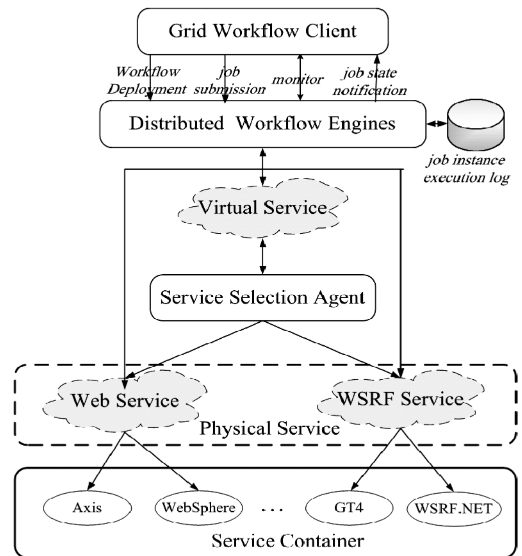

图 1 ServiceFlow 系统体系结构

在 ServiceFlow 中，能够执行原子任务的网格服务(Web 服务和 WSRF 服务)被称为物理服务(Physical Service)，其部署于某个网格节点上，具备真实的服务执行能力. 与物理服务相对应，本系统使用了虚拟服务(Virtual Service)的概念，其与面向对象思想中的抽象类和接口的概念有相似之处. 虚拟服务是对一组有相同接口和服务功能的物理服务的抽象，它本身并不具备服务执行能力. 属于同一虚拟服务的多个物理服务可能因为软硬件环境、执行时间、价格、安全性、可靠性和信誉值等差异而具有不同的服务质量(Quality of Service，QoS). 本系统支持虚拟服务作为原子服务参与构建服务流程；同时，在工作流作业执行过程中，当遇到的原子服务是由虚拟服务承担时，动态服务选择代理根据服务匹配和选择策略，进行虚拟服务到物理服务的映射选择和动态绑定，即从属于同一虚拟服务的多个物理服务中，选择 QoS 综合值最优的物理服务进行作业分派.

服务容器是支持服务实例的运行时环境，核心是 SOAP 引擎，能够为各个服务提供实例启动、消息传递、服务调用、服务安全等保障. 本系统支持两种类型的服务容器：一种是标准 Web 服务(Web Service)容器，如 Apache 的 Axis；另一种是 WSRF(Web Service Resource Framework)服务容器，如 Globus 联盟的 GT4，Microsoft 的 WSRF.NET 等.

1.2　分布式工作流引擎

分布式工作流引擎是整个系统的核心部分，如图 2 所示，其主要分为 3 层：对外接口层、执行管理层、工作流引擎层.

对外接口层提供与系统客户端的交互. 作业队列子模块(Job Queue)接收用户提交的作业请求. 对于工作流作业，我们对 JSDL(Job Submission Description Language)进行了扩展，即通过 WJSDL 来描述作业的 I/O 参数和软件硬件执行环境(软件版本、CPU 频率和个数、磁盘空间、可用内存等)，还增加了对 QoS 请求的描述(如执行时间、可靠性、安全等级和信誉等). 作业解析子模块(WJSDL Parser)负责将 WJSDL 定义的作业请求进行解析，使之成为能够被系统理解的可调度和执行的作业对象. 监控子模块(Monitor)负责对引擎、服务、作业实施分级监视和控制，如对作业实施启动、挂起、恢复、销毁、中止等操作. 复合服务部署子模块(Deployment)负责将已定义好的复合服务部署到工作流引擎上.

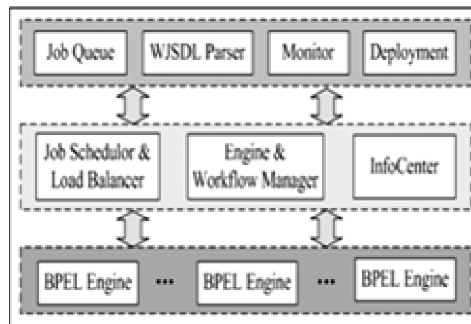

图 2　分布式工作流引擎逻辑结构

执行管理层主要功能是对引擎、复合服务、工作流作业进行管理. 作业调度与负载均衡器子模块(Job Schedulor and Load Balancer)负责从作业队列中提取作业请求并解析，然后结合各个引擎实时负载，依据作业负载均衡调度算法，在多个工作流引擎之间进行选择和作业分派. 引擎和工作流管理器子模块(Engine and Workflow Manager)负责对各个引擎、服务和作业进行直接管理和控制，具体包括：将复合服务部署到各个引擎；对引擎、服务、作业实施分级控制操作，记录并更新它们的注册信息、状态信息；最后返回作业结果等. 信息中心子模块(InfoCenter)负责记录引擎、服务、作业的注册信息和状态信息，记录执行日志等.

工作流引擎层(BPEL Engine)提供复合服务和工作流作业的运行环境，包括复合服务定义解析、工作流作业实例创建和维护、在参与的原子服务之间传递消息、服务状态的监控和通知、容错和补偿、安全和信任机制等.

在每个工作流引擎环境中，本系统设计并部署了一个基于心跳检测机制的状态传感器，每隔一段时间便主动探测引擎、复合服务、工作流作业、物理资源(CPU 和内存使用率)的状态. 当它们的状态改变时，由传感器主动报告，信息中心适时更新状态记录.

2　工作流作业调度

2.1　作业排队模型

本系统采用了分布式多引擎架构和复合服务多点部署策略，同一复合服务被部署在多个正常运行的引擎中. 工作流作业请求提交后，排队等待和被调度到某个引擎的复合服务上执行的过程，本质上构成一个排队模型.

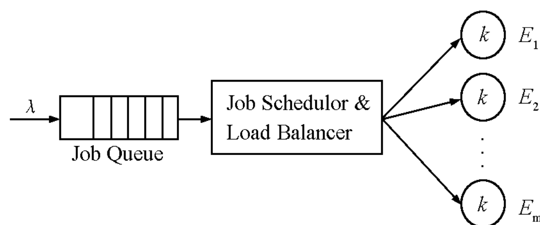

图 3　工作流作业排队模型

设工作流作业到达速率(单位时间内到达的作业数目)服从无记忆的泊松过程，作业的执行时间服从指数分布，作业执行按照先来先服务(FCFS)规则，可执行作业的引擎数目为 m，则作业排队等待和被调度执行的过程近似为一个 $M/M/m$ 排队模型. 如图 3 所示，与传统 $M/M/m$ 排队模型每个服务员同时只能为一个客户服务不同，本系统安装工作流引擎的软硬件环境同构(CPU、内存、OS 等相同)，而每个引擎上可同时执行的最大作业数目为 $k(k\geqslant 1)$，故我们称该模型为 $M/M/m\text{-}k$ 排队模型.

标记作业到达速率为 λ，每个引擎的作业执行速率为 μ(每个作业的平均执行时间为 $T_j=1/\mu$)，则任何一个引擎的耗用率为 $\rho=\lambda/(m\mu)$.

为简单起见，假设 k 为 1，从理论上可推得 $M/M/m$ 模型系统的一些性能预期. 在某一个时刻，系统空闲(没有作业被执行)的概率 P_0 为

$$P_0=\left\{\left[\sum_{i=0}^{m-1}\frac{(m\rho)^i}{i!}\right]+\left[\frac{(m\rho)^m}{m!\ (1-\rho)}\right]\right\}^{-1} \tag{1}$$

而在某一时刻，系统中有 n 个作业的概率 P_n 为

$$P_n=\begin{cases}P_0\left(\dfrac{\lambda}{\mu}\right)^n\dfrac{1}{n!} & n<m\\ P_0\left(\dfrac{\lambda}{\mu}\right)^n\dfrac{1}{m!\ m^{n-m}} & n\geqslant m\end{cases} \tag{2}$$

那么，系统中的平均作业数目 N(包括排队等待的作业和正在被执行的作业)，

$$N=\sum_{n=0}^{\infty}nP_n=m\rho+\frac{\rho}{1-\rho}\frac{(m\rho)^mP_0}{m!\ (1-\rho)} \tag{3}$$

作业响应时间(即从作业提交到执行结束所花费的总时间包括排队等待时间和作业执行时间)为

$$T=\frac{N}{\lambda}=\frac{1}{\lambda}\left[m\rho+\frac{\rho}{1-\rho}\cdot\frac{(m\rho)^mP_0}{m!\ (1-\rho)}\right] \tag{4}$$

其中，每个作业平均排队时间

$$T_q=T-\frac{1}{\mu}=\frac{1}{\lambda}\cdot\frac{\rho}{1-\rho}\cdot\frac{(m\rho)^mP_0}{m!\ (1-\rho)} \tag{5}$$

由此，系统中的作业平均排队长度

$$N_q=\lambda T_q=\frac{\rho}{1-\rho}\cdot\frac{(m\rho)^mP_0}{m!\ (1-\rho)} \tag{6}$$

结合公式(4)和(5)，作业响应时间 T 由作业排队时间 T_q 和作业执行时间 T_j 两部分组成. 当 T_j 比较固定时，设法减少 T_q 就显得非常重要. 在 λ 和 μ 一定的情况下，增大 m，即增加引擎的数目，可以减少作业排队长度 N_q 和作业排队时间 T_q，从而降低作业响应时间 T，并且可以提高系统负载能力和作业吞吐率，降低单个引擎资源的耗用率 ρ.

2.2 负载均衡调度算法

在实际系统中，虽然单个引擎可同时承载多个作业实例执行，但若作业数目过多，将严重影响引擎执行效率. 因而，必须根据支撑引擎的软硬件环境设置适当的作业数目阀值 k. 另外，在 m 个引擎之间，为了防止作业被集中调度，本文基于 2.1 节提出的作业排队模型设计了 3 个负载均衡引擎选择与作业调度算法：轮流调度算法(Round Robin Scheduling Algorithm，RRSA)、随机调度算法(Random Scheduling Algorithm，RSA)和最少作业数目调度算法(Minimal Job Sum Scheduling Algorithm，MinJSSA).

3 个算法所共有的思想是：首先，筛选可调度作业的候选引擎列表(Candidate Engines List，L_{CE}). 即从所有引擎列表(Engine List，L_E)中逐个查询引擎(Engine，E_i)，判断其运行状态是否正常且正在执行的作业数目是否少于 k，若条件满足，则将该引擎加入 L_{CE}. 然后，根据引擎选取策略进行作业调度. 即若 L_{CE} 不为空且作业等待队列(Workflow Job Queue，Q_J)不为空，则从 Q_J 头部取一个工作流作业(Workflow Job，J_W)，从 L_{CE} 中选择一个引擎(Workflow Engine，E_W)，并把 J_W 分派调度到 E_W 中的复合服务(Composite Service，S_C)上执行. 依次循环轮流调度，直到 Q_J 为空(无作业排队等待)或 L_{CE} 为空(所有引擎上作业数均达到阀值).

轮流调度算法(RRSA)的特点在于：从 L_{CE} 头部选取引擎 E_W，然后分派作业 J_W 到 E_W，若此时 J_W 上的作业数目达到最大值 k，则从 L_{CE} 删除 E_W，否则，把 E_W 加入到 L_{CE} 的末尾. 如此循环调度.

算法 1. 轮流调度算法(RRSA)

输入：作业排队队列 Q_J

所有工作流引擎列表 L_E

输出：Q_J 中的作业被依次提取并轮流调度到可执行作业的引擎上执行

1. for each engine E_i in engine list L_E do

2．if（E_i．status＝＝running &&
3．E_i．runningJobSum ＜k）then
4．L_{CE}← E_i；
5．end if
6．end for
7．while（L_{CE}．size ！＝0 && Q_J．size ！＝0）do
8．get a job J_W from the head of Q_J；
9．get a engine E_W from the head of L_{CE}；
10．schedule Q_J to S_C on E_W to execute；
11．delete J_W from Q_J；
12．Q_J．size −−；
13．E_W．runningJobSum＋＋；
14．if（E_W．runningJobSum＝＝k）then
15．delete E_W from L_{CE}；
16．L_{CE}．size −−；
17．else move E_W to the tail of L_{CE}；
18．end while
19．goto 1．to take the next turn.

随机调度算法（RSA）的特点在于：从 L_{CE} 中利用随机函数 random（L_{CE}）选取引擎 E_W，然后分派作业 J_W 到 E_W．若此时 J_W 上的作业数目到达最大值 k，则从 L_{CE} 中删除 E_W．如此循环调度.

算法 2．随机调度算法（RSA）
输入：作业排队队列 Q_J
所有工作流引擎列表 L_E
输出：Q_J 中的作业被依次提取并随机调度到可执行作业的引擎上执行
1．for each engine E_i in engine list L_E do
2．if（E_i．status＝＝running &&
3．E_i．runningJobSum ＜k）then
4．L_{CE}← E_i；
5．end if
6．end for
7．while（L_{CE}．size ！＝0 && Q_J．size ！＝0）do
8．get a job J_W from the head of Q_J；
9．get a engine E_W＝random（L_{CE}）；
10．schedule Q_J to S_C on E_W to execute；
11．delete J_W from Q_J；
12．Q_J．size −−；
13．E_W．runningJobSum＋＋；
14．if（E_W．runningJobSum＝＝k）then
15．delete E_W from L_{CE}；
16．L_{CE}．size −−；
17．endif
18．end while
19．goto 1．to take the next turn.

最少作业实例数目调度算法（MinJSSA）的特点在于：从 L_{CE} 中选目前承担作业实例数目最少的引擎

E_W，然后分派作业 J_W 到 E_W．若此时 J_W 上的作业数目到达最大值 k，则从 L_{CE} 中删除 E_W．如此循环调度．

算法 3．最少作业数目调度算法(MinJSSA)

输入：作业排队队列 Q_J

所有工作流引擎列表 L_E

输出：Q_J 中的作业被依次提取并调度到目前作业数目最少的引擎上执行

1．for each engine E_i in engine list L_E do
2．if (E_i．status==running &&
3．E_i．runningJobSum <k) then
4．$L_{CE} \leftarrow E_i$；
5．end if
6．end for
7．while(L_{CE}．size！=0 && Q_J．size！=0) do
8．get a job J_W from the head of Q_J；
9．get a engine E_W=minimal Jobs(L_{CE})；
10．schedule Q_J to S_C on E_W to execute；
11．delete J_W from Q_J；
12．Q_J．size --；
13．E_W．runningJobSum++；
14．if (E_W．runningJobSum==k) then
15．delete E_W from L_{CE}；
16．L_{CE}．size --；
17．endif
18．end while
19．goto 1．to take the next turn.

以上 3 个算法的第一步均为检查各引擎状态和其上正在运行的作业数目，生成可选的引擎列表 L_{CE}．该过程花费时间和引擎数 m 有关．在引擎选择阶段，RRSA 采用轮流选择法，所用时间最少；而 RSA 的时间消耗和随机函数相关；MinJSSA 通过一次简单选择排序得到当前作业数目最少的引擎，需要进行 $m-1$ 次比较．总的来说，3 个算法总的时间复杂度均为 $O(m)$，其中 RSA 和 RRSA 算法时间开销较小，虽然 MinJSSA 算法时间开销稍长，但其每次选择的是负载最小的引擎，因而负载均衡效果最佳.

3 性能评价

3.1 测试环境

本系统的测试环境搭建在一个有 36 个节点的集群系统上，节点之间通过 100Mbps Fast Ethernet 互连，每个节点配置了 PIII 1GHz 的 CPU 和 512MB 的内存，操作系统为 Red Hat Linux 9.0．其中 4 个节点用来安装 ServiceFlow，32 个节点安装服务容器(Axis1.2 和 GT4)．

测试用例是一个医学图像流程处理应用，每个处理算法程序被封装为网格原子服务部署在服务容器中，整个流程由原子服务构成的复合服务(Image Service)来表现．

3.2 测试结果及分析

在上述软硬件环境下，经过测试，一个引擎可同时执行的工作流作业数目阀值约为 50．为了保证一定服务质量，设置 $k=40$，并借助 JMeter 按一定时间间隔发送作业请求来模拟作业到达，同时通过收集日志来获取实验数据，分别从作业响应时间 T、作业吞吐率 TP、引擎耗用率 U 和执行成功率 S 4 方面对系统进行测试．

作业吞吐率 TP 是指单位时间系统执行完成的作业数目．图 4、图 5 分别给出了在不同作业到达速率 λ 下，单引擎和三引擎的系统的作业平均响应时间和吞吐率．可以看到，单引擎时，随着 λ 增大，T 急剧增

加，而 TP 急剧降低，其主要原因是引擎处于满负荷运行状态，单个作业实例所获得被执行的资源(CPU、内存等)有限，延长了作业执行时间 T_j，另外，新到达作业得不到及时调度和执行，使得排队等待时间 T_q 变长．三引擎($m=3$)系统时，在引擎之间进行了负载均衡调度，故 T 和 TP 都优于单引擎系统．随着 λ 加快，3 个调度算法中 MinJSSA 最优，RRSA 次之，RSA 稍差．

在 $M/M/m$ 理论模型下，用 $\rho=\lambda/(m\mu)$ 来表示引擎的耗用率，反映了引擎的繁忙程度．而在真实系统环境中，工作流作业被执行时，主要消耗支撑引擎的物理资源为 CPU 和内存．因此，我们综合 CPU 和内存的使用率来衡量引擎平均耗用率 U：

$$U=\overline{U_{\text{CPU}}}\cdot w_{\text{CPU}}+\overline{U_{\text{MEM}}}\cdot w_{\text{MEM}}$$
$$w_{\text{CPU}}+w_{\text{MEM}}=1 \tag{7}$$

其中 w_{cpu} 和 w_{mem} 分别为 CPU 和内存的权重．在本试验中，由于医学图像计算工作流更多依赖于计算资源，我们取 $w_{\text{cpu}}=0.6$，$w_{\text{mem}}=0.4$．图 6 给出了在不同作业到达速率 λ 下，单引擎、双引擎和三引擎系统(运用 MinJSSA 调度算法)中引擎平均耗用率 U．可以看出，随着 λ 加快，单引擎系统的耗用率上升很快，更容易接近满负荷工作，双引擎次之，三引擎系统具有相对较低的耗用率．

作业执行成功率 S 在某种程度上能够反映系统的可依赖性，为一段时间内执行成功的作业数 N_{suc} 和执行的总作业数 N_{total} 的比率．从图 7 可以看出，在不同 λ 下，三引擎系统的作业成功率始终最高，双引擎次之，单引擎要差一些．

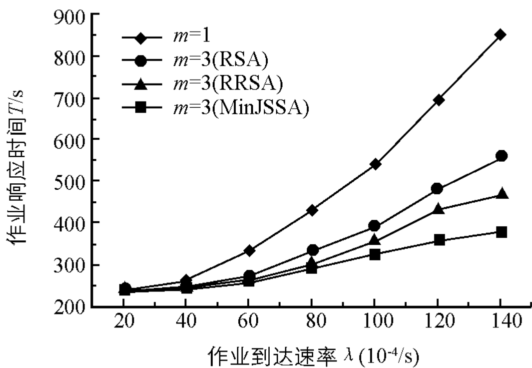

图 4　作业响应时间 T($k=40$)

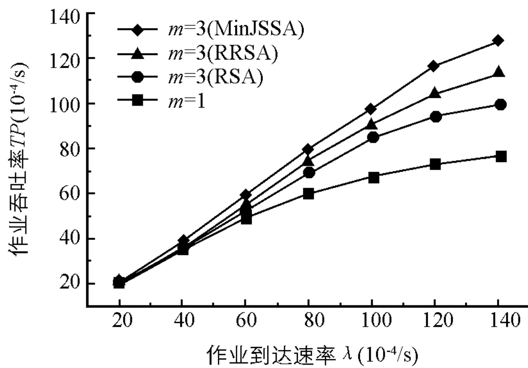

图 5　作业吞吐率 TP($k=40$)

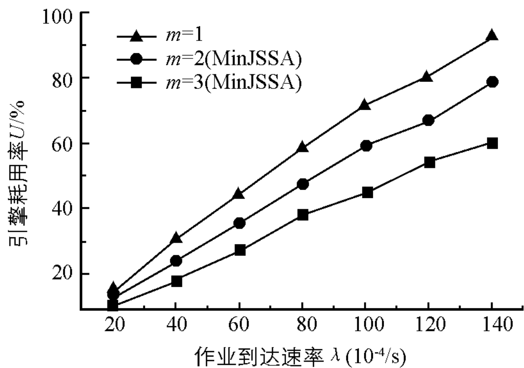

图 6　工作流引擎耗用率 U($k=40$)

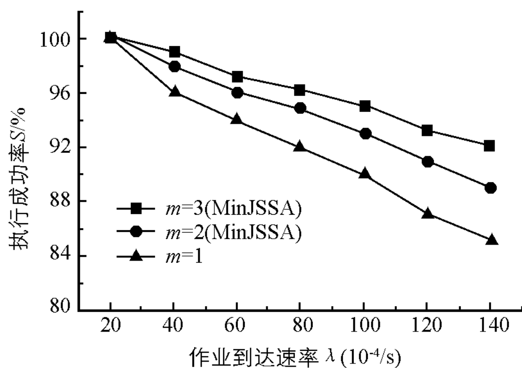

图 7　作业执行成功率 S($k=40$)

另外，引擎是工作流管理系统的核心，其可靠性决定了整个系统的可靠程度．单引擎往往容易造成负载瓶颈和单一失效点．在分布式多引擎架构下，除了可以提高系统负载能力，其整个系统的可扩展性和可靠性也大大增强．设单个工作流引擎的可靠性为 R，则在集中式单引擎模型中，引擎子系统的可靠性就是 R，而分布式多引擎模型下引擎部分是由 m 个同构的单引擎构成的并联系统，其可靠性理论上为

$$R_m=1-\prod_{i=1}^{m}(1-R) \tag{8}$$

若某个时间单个工作流引擎的可靠性 R 为0.8，根据公式(8)，由3个同构引擎构成的分布式系统中引擎子系统的可靠性就可以达到0.992.

4 总 结

研究了面向服务的网格工作流管理系统，扩展了服务流程定义BPEL语言，支持将Web服务、WSRF服务和虚拟服务作为原子服务参与构建复合服务；提出了分布式多引擎系统架构；基于 $M/M/m$ 排队模型，设计了负载均衡调度算法，在各引擎之间进行作业优化调度. 实验结果表明，与集中式单引擎系统相比，分布式多引擎系统减少了作业平均响应时间，增强了系统负载能力和可靠性，提高了作业吞吐率和执行成功率，同时具备良好的可扩展性.

参考文献：

[1] 周世杰，秦志光，刘锦德. 工作流管理系统互操作技术研究 [J]. 电子科技大学学报：自然科学版，2002，31(2)：145－150.
[2] 祁正华，任勋益，王汝传. 大规模电力网格体系结构 [J]. 重庆邮电大学学报：自然科学版，2006，18(3)：393－396.
[3] CAO J，JARVIS S，SAINI S，et al. Workflow Managament for Grod Computing [C] //Proceedings of the 3rd International Symposium on Cluster Computing and the Grid. New York：IEEE Computer Society Press，2003：198－205.
[4] 陈 峰，荣晓慧，邓 攀，等. 设备协同技术及其系统软件研究综述 [J]. 电子学报. 2011，39(2)：440－447.
[5] 应 宏. 网格系统的组成与体系结构分析 [J]. 西南师范大学学报：自然科学版，2004，2(4)：586－590.
[6] 李金忠. 基于AGWL的网格工作流规范研究及其应用 [J]. 井冈山学院学报：自然科学版，2008，29(3)：30－32.
[7] 苑迎春，李小平，王 茜，等. 基于逆向分层的网格工作流调度算法 [J]. 计算机学报，2008，31(2)：282－290.
[8] 刘 洋，桂小林，徐玉文. 网格工作流中基于优先级的调度方法研究 [J]. 西安交通大学学报：自然科学版，2006，40(4)：411－414.

A Grid Workflow Management System Based on Distributed Multi-Engine Framework

ZHAO Gang

Xi'an Aeronautical Polytechnic Institute，Xi'an 710089，China

Abstract：Currently most grid workflow systems are built in a centralized single engine with low workload capacity，single point of failure and poor scalable performance. This paper describes ServiceFlow，a grid service-oriented workflow management system，which extends BPEL to support Web service，WSRF service and Virtual service participating in grid service composition，and adopts a distributed multi-engine framework，in which，based on $M/M/m$ queuing model，load-balancing scheduling algorithms are proposed to dispatch jobs among engines. The results of performance evaluation show that ServiceFlow can achieve shorter mean job response time，higher throughput and better dynamic scalability compared to the centralized single-engine workflow system.

Key words：grid workflow；$M/M/m$ queuing model；load balancing；job scheduling

责任编辑 张 枸