一种分布式工作流中基于负载平衡的调度算法*)

张 坚 刘春林 谭庆平
(国防科技大学计算机学院 长沙 410073)

摘 要 工作流管理系统(Workflow Management System)帮助企业实现分布式异质环境中的人工和自动任务的全面流程化。本文提出一种新的分布式 WFMS 负载平衡调度算法,解决单节点引擎负载过重的问题。改进后的负载平衡技术比轮询调度算法更能保证多引擎间负载基本均衡。当多引擎间任务调度出现不平衡,导致某些节点负载过重或是某个节点出现故障时,多引擎能够根据相应模型策略和实际的信息来动态调整各节点的负载,这样也可以在一定程度上解决有大量分布式事务处理时的性能问题。另外,负载指数调度算法实现简单,尽可能地减少了引擎负担。

关键词 工作流引擎,负载平衡,负载指数

A Scheduling Algorithm Based on Load Balancing in Distributed Workflow

ZHANG Jian LIU Chun-Lin TAN Qing-Ping
(School of Computer, National University of Defence Technology, Changsha 410073)

Abstract The enterprise business requires the flow process integrating manual and automatic tasks by WFMS(Work-Flow Management System). This paper introduces a new method about load-balancing for distributed WFMS, which resolves over-load balancing in single site. This method has been compared with Round-Robin Scheduling and the experiment result shows that the new methods designed can assure the load-balance in muti-engine more efficiently. As scheduler leads load unbalance, the multiengine adjusts it among many engines based on appropriate model tactics and information in real time, which advance the efficiency of large scale distributed businesses. In addition, the Load Index Scheduling has simple realization, lessening the engine load as possible.

Keywords Workflow engine, Load balance, Load index

1 引言

近年来,随着 Internet 技术的发展,BPR(Business Process Reengineering)领域出现了一种新的支持网络计算的业务处理方式——工作流管理系统(workflow management system,简称 WFMS)。WFMS 把人工业务或异构环境中的自动应用系统集成到统一的业务过程中,它利用企业已有的计算设施,通过业务任务间的依赖关系进行复杂的任务协同调度。我们的目标就是要在性能可测量的情况下保证这样一个 WFMS 的工作负载平衡。

平衡一个普通分布式系统的工作负载需要做到以下几点工作:1)通过合适的指数来描述工作负载等级。2)获取当前有效过程单元的负载信息。3)为满足预先定义的性能标准,设计一个在过程单元中分配工作的调度策略。4)适当地协调负载平衡组件和过程单元,保证在负载平衡策略发生变化时系统结构的稳定。

2 工作流概述

工作流是全部或部分由计算机支持或自动处理的业务过程。工作流的最根本目的是在正确的顺序下,各项任务由最合适的人员执行,使得业务过程的执行达到最大效率。工作流引擎是工作流管理系统的核心。工作流引擎是为工作流管理系统在定义时提供支持,同时在运行时提供解释和执行服务的一组数据模型和软件。调度模型是以过程模型为核心,和其他模型一起共同组成的模型系统。各模型之间相互独立又相互关联,调度模型反映了各模型之间的调用关系。高效的调度模型可以加快调度速度,适合于大规模工作流系统的建模和调度。

3 工作流中负载平衡技术

如何采取有效的调度策略来平衡多结点(机)的负载,从而提高整个系统资源的利用率,已成为当前研究热点。在下面的讨论中,我们假定每个结点上的任务是动态产生的,每个结点的负载大小是动态变化的,因而不考虑静态负载平衡调度方法,只考虑动态负载平衡调度策略。

3.1 工作流负载信息参数

当客户访问多资源时,任务提交的时间和所消耗的计算资源千差万别,它依赖于很多因素,包括任务请求的服务类型、当前网络的情况以及当前服务器资源利用的情况等等。负载平衡导致客户长时间的等待,使整体的服务质量下降,因而有必要采用一种机制,使得控制器能够实时地了解各个结点的负载状况,并能根据负载的变化做出调整。而现在引擎中一般采用的轮询调度算法还不能很好地解决这个问题。在这个时候保证多个引擎间的负载平衡就显得比较重要了。

在考虑分布式工作流系统中的负载平衡前,我们要明确什么是 WFMS 工作负载,并考虑如何通过负载指数来描述工作负载,以及如何从工作流引擎中收集负载信息。WFMS 主要目的就是为了自动化商业过程。因此,WFMS 的基本工作负载就是商业过程。而商业过程又包括各种商务活动。一个商业过程的执行时间也包括了所有相应活动的执行时间。WFMS 可选的负载参数包括处理执行时间、活动处理实例序号和二个邻近活动的工作流引擎延迟等等。WFMS 中商业过程的处理时间无论对于客户还是 WFMS 都是一个非常重要的性能参数。如果是多处理实例同时进行时,这些实例就会共享 WFMS 资源。当实例序号增加时,工作流引擎中每个处理实例的共享处理时间在一定时间段内也会增加。我们可

以对参数进行如下描述：

定义 1 Wn_i 表示过程 P 中一个活动，$i=0,1,\cdots,n-1$

每个活动 Wn，$Wn\in P$，且处理过程 $P=\{Wn_i \mid i=0,1,\cdots,n-1\}$

定义 2 Tp 表示是商业过程实例 P 的执行时间

定义 3 Twn 表示过程 P 中前一个活动 Wn_i 的开始时间和当前活动 Wnj 的开始时间的间隔时间，并且

$Tp=\sum Twn$

定义 4 Tre 表示一个活动 Wn_i 的资源角色执行时间，Ten 表示同样活动 Wn_i 的引擎执行时间，并且

$Twn=Tre+Ten$

对于这个过程的特定实例，执行部分包括 m 个活动。由于执行部分的一些活动有可能因为过程中的循环会执行多次，因此 m 是有可能大于 n 的。如果活动过程实例数量增加，Tp 就会增加。Tp 在一定程度上反映了工作负载等级的倾向。如果仅以 WFMS 的单个引擎或分布式 WFMS 的多引擎来衡量负载等级，Tp 可以是 WFMS 负载指数。

3.2 工作流负载平衡调度模型

通过运行一个简单的商务过程测试 WFMS，我们在执行活动时观察各种时间参数。测试工作是在 Window2000 操作系统平台上进行的。图 1 说明不同过程实例这些参数的不同倾向。在测试中，发起者以 400 过程实例/每小时启动 300 个过程实例。这些过程所有的资源都是 java 客户端程序。java 客户端检查工作域并通过实践参数改变一些域值。对于每一个过程实例，第一个活动客户通过调用实例活动发起者来初始化活动。其中记录了活动(process)发起信息。第二个客户端响应第一个活动。其检查发起者的请求并计算时间戳。第三个客户端响应第二个活动，同时也检查发起者的请求并计算时间戳。图中表明了测试活动的前 40 个活动实例前 n 个活动(T_i,T_j)的 Twn。T_i($I=1,2,3$)代表发起者执行的延迟，T_1 代表第一个活动执行的延迟。

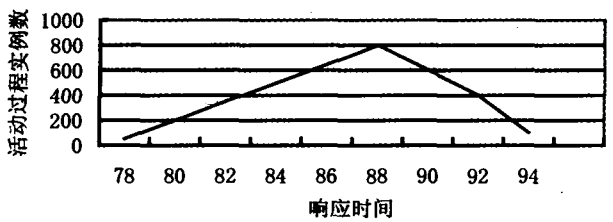

图 1 Twn 和活动过程实例数之间的关系

图 1 说明了 Twn 和活动过程实例数之间的关系。测试中一共有 1000 个过程实例。其包括三个部分。首先在起始点开始的 Twn，当实例到达频率一定时，处理过程实例数也是基本呈直线上升。在时间到 88 时，所有 1000 个过程实例都已经开始处理。从这一点开始就不会有新的任务加入了。中间部分说明了在相同时间段内，活动过程实例数的变化。活动过程实例数最多的是 800 个。此时时间是 88。这就意味着 Twn 可以准确地表示负载增加倾向。到 88 时，再没有新的过程实例加入，Twn 开始减少。到 92 时，$WFMS$ 完成 1000 个过程实例中的第一个活动。这也说明可以准确地表示负载减少倾向。通过此图，我们定义描述曲线的公式如下：

$$Twn=N*a+H \tag{1}$$

其中 N 表示活动过程实例数，A 为执行响应时间增加率，H 为固定容量常数，a、H 都是性能参数(a 表示工作引擎处理能力函数，H 表示一个平台的处理容量)。由这个图和通过这个图所得到的公式说明用 Twn 可以比较好地表达一个引擎的负载情况。基于这些信息，我们为 WFMS 引入一个概念：L 表示负载指数。我们定义负载指数 L 的公式如下：

$$L=\sum \text{Twn-engine}/k \tag{2}$$

Twn-engine 表示不同引擎的 Twn，k 表示在一段时间内所有活动过程实例中完成的活动数。当系统刚开始时，L 的初始值设为 0。如果在设定的时间段内没有活动完成，L 就保持其最后的值。

有了负载指数就可以通过其进行任务分配。工作流引擎中的控制器则根据各个节点的 L 值来分配任务。L 值越低的节点，则越先分到任务。

3.2 工作流负载指数调度算法

根据前面所提出的负载指数以及负载平衡的思想，我们提出一个新的调度算法。该算法的主要思想是通过负载指数作为结点负载参考，调度器根据负载指数来判断当前各个结点负载状况，然后将任务分配给各个结点中负载指数最小的那个结点。

我们考虑单个调度器，每个调度器中都有唯一的一个调度进程。每个调度进程主要循环不断地从局部于它的消息队列里取出消息，然后根据不同的消息类型采取相应的操作。在以下算法中，统一用 send(*,*,*,*,*)表示一次消息传递。其中，前两个参数分别为该次消息传递的源进程节点和目的进程节点。第 3 个参数不为空则表示任务实例进程 W，如果第 3 个参数为空则说明是返回消息。第 4、5 个参数分别为本节点和目标节点的负载指数，在算法中用 L 表示负载指数，H 表示节点固定容量常数。

调度进程 S 所收到的任务消息 w 有两个来源：本结点上的任务实例进程，或者是邻接结点上的调度进程。

算法 4.1 负载指数调度算法

```
输入：本节点任务实例；
输出：执行该任务队列的节点
 1 if 当前任务消息来自本地结点上的任务实例进程 A then
   {
     1.1 检查本节点的 vecload(S)值，
     1.2 if vecload(S)< H then
         {
             Ws =Ws∪{w}，算法结束；
         }
     1.3 else
           1.3.1 遍历 Cs 中的每个节点，得到 vecload()的最小值 L′
                 和对应节点 S′
           1.3.2 执行 send(S,S′,W,L,L′)，算法结束；
   }
 2 else if 任务消息 w 来自邻接结点上的调度进程 S then
   {
     2.1 检查本节点的 vecload(S)值
     2.2 if vecload(S)< H，then
         {
             Ws =Ws∪{w}，算法结束；
         }
     2.3 else
           2.3.1 遍历 Cs 中的每个节点，得到 vecload()的最小值 L′
                 和对应节点 S′，2.3.2 执行 send(S,S″,W,L′,L″)和
                 send(S′,S,NULL,L′,L)
           2.3.2 算法结束；
   }
```

由上述算法可知，调度进程 S 首先检查是否能把任务 w 送给负载指数小于 H 的本结点，从而减少消息传递的消耗，若行，则将 w 放在本结点的未计算任务队列 Ws 中。否则进一步检查本结点上负载指数数组，把任务送到负载指数最小的邻接结点上。收到其他结点传递来任务的结点首先取出消息中的参数 L'和 L''，将参数 L'更新到本地节点的负载指数数组中，再拿从其他节点获得的本地负载指数 L''与本地节点数组中的负载指数进行比较，从而判断其他节点发来的负载信息与本节点的是否一致。这些检查非常关键，它关系到如何最快最有效地获得负载平衡，又能同时更新各个节点的负

载指数数组。如果每次调度后 S 的负载状态与其他节点发来的信息基本一致，那么将 w 就放在本结点的未计算任务队列 Ws 中。否则就重新检查本地结点上负载指数数组，把任务送到负载指数最小的邻接结点上，并且将本地节点负载指数参数信息发送给消息源节点，从而更新源节点的负载指数数组。

算法的优点：

1. 采用该调度算法的调度器开销恒定（与当前系统负载无关），实时性能更好。

2. 能及时反馈各个结点的负载情况，并根据负载信息调整任务分配。

3. 算法实现简单，很大程度上减轻了工作流引擎的负担。

3 实验比较

3.1 分布式工作流引擎中调度算法实现

我们在现有的工作流平台基础上，主要是对调度器进行设计从而实现该调度算法。如第 3 节中所提到的调度器部分，我们对其进行改进设计。调度器的类图如图 2。

• Engine：普通类，向外部系统提供启动引擎，停止引擎，暂停引擎和重新唤醒引擎的 API，分别维持对 Scheduler 和 NodeQueue 两个类的引用；

• NodeQueue：普通类，主要负责维持一个待执行节点的队列，通过 java. util. LinkedList 可构成一个简单的队列。这个类提供了向队列中添加、删除和查找特定节点 API，以及轮询队列是否为空和从队列取出特定节点的方法；

• Scheduler：实现 java. lang. Runnable 接口，通过这种可以比较方便地实现重用线程（reuseable-thread），这个类的主要功能是判断待执行节点队列是否为空，不空则取出第一个节点执行之，并且通过调用 EngineLoad 类实现负载指数调度算法。

• EngineLoad：提供负载指数函数，这个类的主要功能就是计算出当前节点的负载指数，并提供给 Scheduler 类。

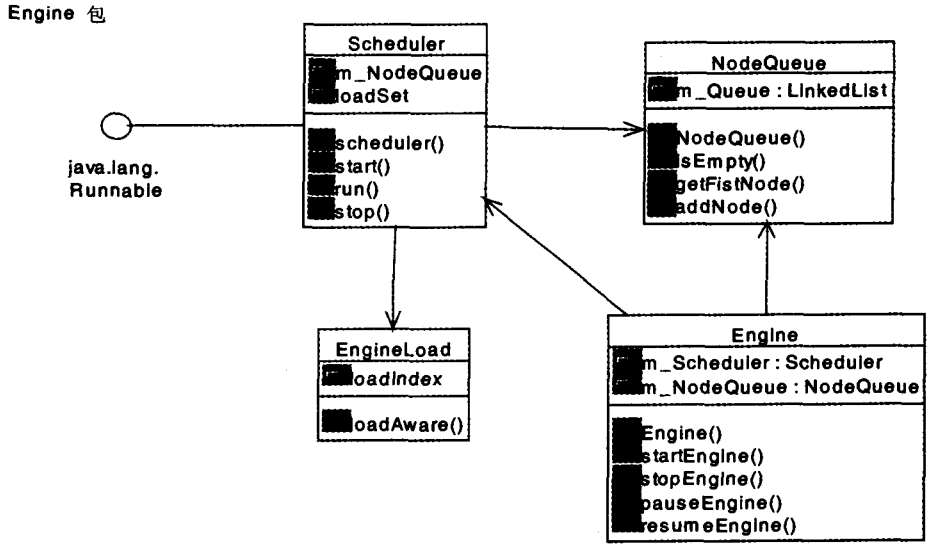

图 2 工作流引擎类图

我们在 Scheduler 类中实现上述调度算法，而将负载指数的计算交给 EngineLoad 类。调度器进行调度时，先通过 EngineLoad 类中的 loadAware() 函数计算获取负载指数，并将计算结果返回给 Scheduler，从而根据 Scheduler 所存的各个节点的负载指数信息来判断任务的调度分配。

3.2 实验结果

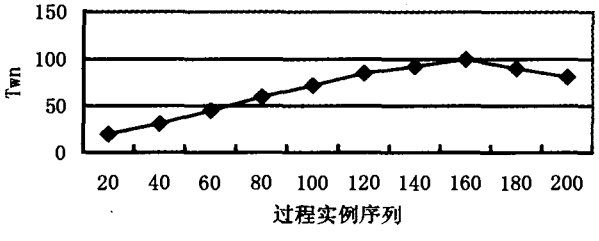

图 3 Twn/过程实例序列关系：轮询调度

我们通过 200 个商务过程的活动反应延迟比较两个不同方法：轮询调度（Round-Robin Scheduling）和负载平衡调度在负载平衡方面的效果。图 3 是轮询调度时 *Twn* 的情况，图 4 是负载平衡调度时 *Twn* 的情况。轮询调度可以保证每个引擎的过程实例数是基本差不多的。图 3 可以看到轮询调度时 *Twn* 基本上是随着到达过程实例的增加而增加，当不再有新的实例到达时，*Twn* 就会随之减少。负载平衡调度则可以保证每个引擎的负载不会相差太远，也就是 *Twn* 不会相差太远。每个任务的完成时间都差不多，这样就保证了任务完成的及时性。图 4 可以看到负载平衡调度时 *Twn* 基本上是保持在 45 上下。

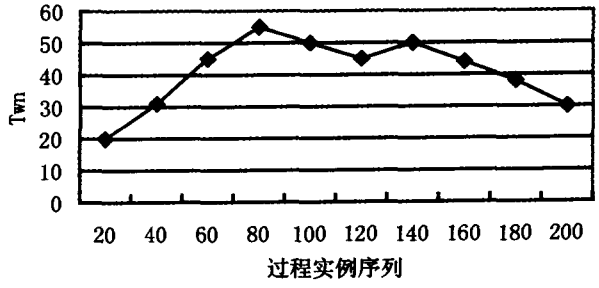

图 4 Twn/过程实例序列关系：负载平衡调度

结论 我们在分布式环境下的多引擎 WFMS 中提出采用负载平衡调度算法，首先通过评估 *Twn*，再依此提出负载指数，最后通过衡量负载指数来达到负载平衡的目的。实验证明我们所提出的负载平衡调度确实要比轮询调度更适合。

基于椭圆曲线的安全组播多层接入控制方案*)

徐守志　谭运猛　杨宗凯　程文青
(华中科技大学电子与信息工程系　武汉 430074)

摘　要　具有多层服务结构的组通信要求有多层接入控制的能力。已有的研究方案在密钥更新时存在开销大、时延长的缺点,使组播的服务质量难以保证。本文提出基于椭圆曲线密钥体制的斜树密钥管理方案,算法只需更新少量密钥就能满足系统的安全需求。与已有方案相比,效率有很大提高。

关键词　安全组播,多层接入控制,椭圆曲线,多层服务

Elliptic Curve Cryptosystem Based Scheme of Hierarchical Access Controlling for Secure Multicast

XU Shou-Zhi　TAN Yun-Meng　YANG Zong-Kai　CHEN Wen-Qing
(Department of Communication Science & Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　Multi-layer services group communications need function of hierarchical access controlling. Existing methods have disadvantages of high cost and long delay, which makes they not assure the quality of service(QoS) of multicast. An inclined tree-group key management scheme based on elliptic curve cryptosystem is presented in this paper, which needs to renew only several node keys to provide perfect secrecy. It is more efficient than previous works.

Keywords　Secure multicast, Hierarchical access control, Elliptic curve cryptosystem, Mmulti-layer services

具有多层服务结构的多媒体组播有广泛的应用前景。这种组播要求具有多层接入控制,即用户加入的等级不同,所享有的服务也不同,等级高的用户享有所有等级低的服务[1]。IETF 提出相关安全标准[2],主要包括:加入用户不能解密加入前的组播数据(后向安全);离开的用户不能获得将来的组密钥(前向安全)。如何安全高效地更新组会话密钥(SK: session key)一直是研究的难题。目前的主要成果包括基于逻辑层次密钥树(LKH: Logical Key Hierarchy)[3]和单向函数树方案(OFT: One-Way Function Tree)[4]。LKH 大多是基于 Diffe-Hallman,需要进行大量的幂指数运算。OFT 应用 Hash 函数以减小通信开销,但存在串通攻击的安全问题。而且这些成果若应用在多层服务的安全组播中,需要对多个子组分别管理,系统的存储开销、更新开销都很大,因此难以有效地应用。

鉴于此,本文提出基于椭圆曲线密钥体制的斜树密钥管理方案,简化算法设计。分析算法的安全性,并对算法的效率进行分析和仿真比较。

1　基于椭圆曲线的组共享密钥

迄今的研究表明,椭圆曲线密码体制(Elliptic Curve Crypto system,简称 ECC),除超奇异曲线和异常曲线外,其求解算法都是指数时间算法[5],能以较小的开销实现较高的安全性。

1.1　椭圆曲线 $E_p(a,b)$ 的离散对数难题

定义 1　设 $GF(p)$ 是一个特征 $p\neq 2,3$ 的有限域,$\exists a,b\in DF(p)$ 满足 $4a^3+27b^2\neq 0$,$p\in(2^t,2^{t+1})$,且 p 为一个大素数。取方程:$E: y^2=x^3+ax+b$,则 $E_p(a,b)=\{(x,y)\mid y^2=x^3+ax+b(\mathrm{mod}\ p)\}\cup O$ 按照文[5]定义的零元、逆元、加法+

*)基金项目:国家自然科学基金资助项目(90104033)。徐守志　副研究员,博士生,研究方向为网络安全、应用密码学;杨宗凯　教授,博士生导师,研究方向为现代信息网络理论及其应用和现代数字信号处理技术;谭运猛　博士,副教授,研究方向为电子支付、应用密码学。

轮询调度只能保证每个引擎所完成任务数目是基本相同的,而不能保证每个引擎的负载基本均衡。特别地,当业务过程所需的时间较长时,负载平衡调度的优点更加突出。当多引擎间任务调度出现不平衡,导致某些节点负载过重或是某个节点出现故障时,多引擎能够根据相应模型策略和实际的信息来动态调整各节点的负载,这样也可以在一定程度上解决在有大量分布式事务处理时性能问题。下一步,我们还应该继续完善负载平衡算法,使之根据不同实际应用采用不同的粒度来衡量负载指数,达到大量的简单任务时采用粗粒度来衡量,而对比较复杂的任务采用细粒度来衡量。

参 考 文 献

1　陈华平,陈国梁. 分布式动态负载平衡调度的一个通用模型,1998
2　Baker S, Moon B. Distributed Cooperative WebServers. In: Proc. of the 8th Intl. WorldWide Web Conf. 1999
3　Farrell R. Distributing the Web load. Network World, 1997
4　傅强. 一种适用于机群系统的任务动态调度方法,1999
5　Ferrari D, Zhou S. A load index for dynamic load balancing. In: 1986 Proc. of Fall Joint Computer Conf. Dallas, Texas, Nov. 1986
6　Milojicic D S, Pjevac M. LoadBalacing Survey. In: Proc. of the Autumn 1991 EurOpen Conf.
7　WestBrook J. Load Balancing for Response Time. Journal of Algorithm, 2000, 35: 1～16
8　Barbara D, Mehrotra S. INCAs: managing dynamic workflows in distributed environments. Journal of Database Management, Special Issue on Multidatabases, 1996