一种改进的死锁和活锁避免资源联合分配协议

伍之昂,曹 杰,王有权
(南京财经大学江苏省电子商务重点实验室,江苏南京 210003)

摘 要: 提出一种改进的死锁和活锁避免资源联合分配协议——OODP3(Optimal ODP3),OODP3 基于 ODP3(Order-based Deadlock Prevention Protocol with Parallel requests)的安全状态方法避免死锁和活锁,但是,OODP3 将其时间复杂度降到多项式级,并对 OODP3 的正确性进行了理论证明,实验结果表明 OODP3 的执行速度快,而且比现有的资源联合分配协议具有更优越的性能;最后进一步讨论了退避时间协议和资源分配策略对 OODP3 性能的影响.

关键词: 资源联合分配协议; 死锁; 活锁; NP_complete
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2011) 11-2589-08

An Improved Deadlock and Livelock Free Protocol for Resource Co-Allocation

WU Zhi-ang, CAO Jie, WANG You-quan
(*Jiangsu Provincial Key Laboratory of E-Business, Nanjing University of Finance and Economics, Nanjing, Jiangsu* 210003, *China*)

Abstract: An improved deadlock and livelock free resource co-allocation protocol called OODP3 (Optimal ODP3) is proposed. OODP3 utilizes the safe state method in the framework of ODP3 to avoid deadlock and livelock. However, OODP3 reduces its time complexity to polynomial level and theoretical proof is conducted to verify the validity of OODP3. Experiment results show that OODP3 not only executes faster but also achieves a better performance than the existing deadlock and livelock avoidance protocols. At last, how back-off protocol and resource allocation policy affect the performance of OODP3 is further discussed.

Key words: resource co-allocation protocol; deadlock; livelock; NP_complete

1 引言

网格计算、云计算等网络计算使利用跨站点资源执行应用成为可能,很多应用需要同时获得多个资源的访问权限才能成功执行[1~3].随着网络规模的逐步扩大,各种应用急剧增多,竞争资源现象普遍存在.在为数量极多的应用分配资源时,集中式资源分配需要全局的资源状态信息,这在跨越地理范围极大、资源种类极多的环境下是不可行的[4],因此,各应用需要根据分布式资源联合分配协议申请资源,进而获取所需资源.

在分布式资源分配过程中极易导致死锁和活锁,当并发应用集合中的每个应用占有集合中其它应用所需的资源,且各个应用都不释放已占有的资源时,该组并发应用集合将处于永久阻塞状态,这种现象称为死锁.如果应用资源需求无法得到满足时,释放所占有的资源,则容易反复获得和释放同一资源,却不能进一步获得其他资源,该应用资源需求仍得不到满足,这种现象称为活锁[5].

分布式资源联合分配解决死锁问题通常有两种策略:死锁检测和死锁预防[6].死锁检测是指定期检查并发应用的状态,一旦检测到死锁存在,则放弃一个或多个应用所占用的资源,从而使并发应用集合跳出死锁状态,但是,死锁检测依赖于应用之间的消息传递,而在分布式计算环境中,一个应用无法感知其他的应用的存在,从而无法实现消息传递,因此,死锁检测策略难以应用到分布式计算环境中[7].死锁预防能保证应用根据分布式资源联合分配协议申请资源时不会陷入死锁,各个应用可以独立执行资源联合分配协议,而无视其他应用的状态,因此,死锁预防是完全分布式的,且应用间无需进行消息传递.

综上所述,资源联合分配解决死锁问题的策略决定其只能采用预防的方法来避免死锁和活锁,死锁和活锁预防完全依赖于分布式资源联合分配协议,因此,如何设计一种能有效避免死锁和活锁的高效分布式资源联合分配协议成为一个挑战性问题.Jonghun Park 于 2004 年提出一种死锁和活锁避免的分布式资源联合分配协

收稿日期:2010-05-07;修回日期:2010-12-14
基金项目:国家自然科学基金(No.71072172,No.61103229);教育部"新世纪优秀人才支持计划(No.NCET-07-0411);江苏省教育厅高校自然科学重大基础研究(No.07KJA52004);东南大学江苏省网络与信息安全重点实验室资助项目(No.BM2003201)

议——ODP³(Order-based Deadlock Prevention Protocol with Parallel requests)[8],理论证明 ODP³ 能成功避免死锁和活锁,但是,ODP³ 将其子问题——下一安全状态查找归化为背包问题(Knapsack Problem),背包问题是 NP_complete 的,因此,ODP³ 无法保证在多项式时间内找到最优的下一安全状态,所以,ODP³ 是一种启发式算法.本文改进 ODP³ 协议,重新对 ODP³ 中的子问题——下一安全状态查找建模,提出多项式时间复杂度的下一安全状态查找算法,该算法保证在定义任何目标函数的情况下找到最优解,并进行了完整的理论证明.本文提出的协议称为最优 ODP³ 协议(OODP³,Optimal ODP³),时间复杂度分析和仿真实验都表明 OODP³ 比 ODP³ 更加快速,在资源需求量大、目标状态数量多的情况下,OODP³ 能有效地降低平均等待时间、提高资源利用率.

2 研究现状

国内外学者已经在分布式资源联合分配上做了大量研究工作,取得了很多激动人心的成果,Netto 等将分布式资源联合分配方面的研究总结为四个方面[9]:分布式事务、容错、站点间网络总开销及调度优化.分布式事务研究避免死锁和活锁的资源联合分配协议,因此,本文的工作属于该领域.

早期最著名的避免死锁和活锁资源联合分配协议叫做 ODP²(Order-based Deadlock Prevention Protocol),ODP² 为每个资源定义了全局线性序号,并要求每个应用根据升序逐个申请资源[10].但是,ODP² 仅允许应用定义一种资源需求方案,并且 ODP² 容易导致应用申请资源时拥塞、等待时间过长.为解决 ODP² 的缺陷,Park 提出 ODP³ 协议[8],ODP³ 允许应用定义多种资源需求方案,并保证应用能获得其中的一种资源需求方案所规定的资源类型和数量,且应用不陷入死锁或活锁状态.在效率方面,ODP³ 通过发送并发的资源请求降低应用的等待时间.但是,ODP³ 将其子问题——下一安全状态查找归化为背包问题(KP,Knapsack Problem),ODP³ 子算法是解决背包问题的启发式算法,因此,ODP³ 本身也是指数级时间复杂度的算法.针对 ODP³ 的这一不足,本文避免将下一安全状态查找归化为背包问题,通过提出定理 2 来查找下一安全状态,这一改进将 ODP³ 的时间复杂度降到多项式级.

Azougagh 等人提出 ACT 技术(Availability Check Technique)以降低资源联合分配过程中的冲突,他们将资源联合分配问题类比为哲学家就餐问题,应用不断检测所需资源,直到它所需的所有资源都空闲时,才申请并占用资源[11].ACT 是一种 ODP² 或 ODP³ 协议的补充技术,并非对此两种协议的改进,因此,Azougagh 等人的研究与本文的研究是正交的.

3 问题描述

分布式环境下的资源散布在不同的物理站点上,设共有 M 种资源散布在多个物理站点上,M 种资源表示为 $R=\{R_1,R_2,\cdots,R_M\}$,每个物理站点包含一定数量的 M 种资源中的一种或多种.应用需要同时获得一定数量的多种资源才能完成,并且一个应用通常包含多种资源需求方案,比如,$G_\theta=\{R_1+2R_3,2R_2\}$,表示应用 θ 有两种资源需求方案,第一种需要 1 个 R_1 和 2 个 R_3,第二种需要 2 个 R_2,应用 θ 获得任意一种资源需求方案都能完成.在分布式环境下,同一时刻通常有多个应用同时向多个物理站点申请资源,这些同时申请资源的应用称为并发应用集合,记为 $App=\{\theta_1,\theta_2,\cdots,\theta_K\}$.

应用根据其资源需求向多个物理站点并行发送资源请求,物理站点可能收到多个应用发来的资源请求,它根据可用资源列表和资源管理策略为应用分配一定种类和数量的资源,应用收到各个物理站点为其所分配的资源,如果该应用的某资源需求方案得到满足,该应用开始执行,直到完成.如果分配给该应用的资源无法满足该应用的任意资源需求,该应用将持有已分配的资源,并继续发送并发资源请求以申请剩余所需资源,直到它持有的资源满足它的某一资源需求方案为止.我们将应用持有部分资源,但不能满足其任意资源需求方案的状态称为中间状态.

本文使用有限状态机(FSM,Finite State Machine)对应用申请资源的过程进行形式化,Park 将这种有限状态机定义为 SD-RAS(Single-step Distributed Resource Allocation System),将 SD-RAS 作为资源联合分配的形式化工具有如下两点理由:(1)本文的主要贡献——OODP³ 协议是基于 SD-RAS 的;(2)SD-RAS 能方便地描述应用的多个资源需求方案,并且能记录应用在申请资源过程中的每一个中间状态.

由于文献[8]已经详细介绍了 SD-RAS,同时 SD-RAS 亦非本文的贡献,因此,本节简单介绍基于 SD-RAS 的资源联合分配模型的形式化.设 M_θ 是表示应用 θ 申请资源过程的有限状态机,s_θ^0 是 M_θ 的初始状态,表示应用 θ 未分配任何资源.M_θ 的其他状态 s_θ^i,$i=1,2,\cdots,L_\theta$,表示应用 θ 持有一定数量和种类资源后所处的状态,如果 s_θ^i 中资源满足应用 θ 的某个资源需求方案,则称 s_θ^i 为目标状态,否则,称 s_θ^i 为中间状态.$s_\theta^i(R_j)$ 表示在状态 s_θ^i,应用 θ 持有的第 j 种资源 R_j 的数量,G_θ 表示应用 θ 的资源需求方案集合,即 G_θ 中的每个元素表示 θ 的一种资源需求方案.图 1 给出一个 SD-RAS 的例子,例中的应用 θ 有三种资源需求方案,即 $G_\theta=\{2R_1+2R_2,R_1+R_2+R_3,3R_2+R_4\}$,$s_\theta^8$、$s_\theta^9$ 和 s_θ^{10} 是与此三种需

求方案相对应的目标状态,s_θ^1 到 s_θ^7 是中间状态.死锁是由不当的资源分配顺序引起的,为了预防死锁,SD-RAS 中的一些中间状态是不允许的.为此,文献[8]提出安全状态(safe state)这一概念,安全状态是一个中间状态,应用经过安全状态可以成功达到某一目标状态,而不陷入死锁.在文献[8]中,Park 首先提出安全路径(safe path),基于安全路径提出安全状态的概念,然后提出定理 1 用以判定安全状态,并且证明了定理 1.安全状态定义和定理 1 是本文的研究基础,但并不是本文的贡献,因此,本文只给出安全路径和安全状态的定义,以及定理 1 的内容,不对定理 1 进行证明.

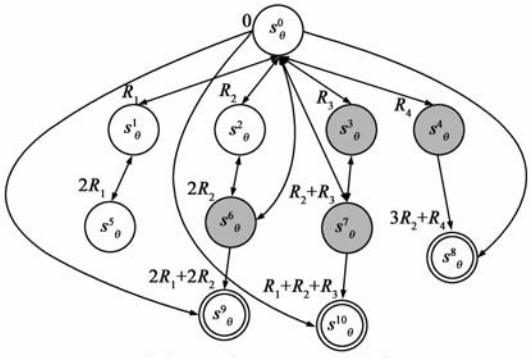


图1　一个SD-RAS的例子

定义 1 安全路径　SD-RAS 中服务实例 θ,如果 M_θ 中一条有限长路径 $P=\langle s_\theta^{i_1}, s_\theta^{i_2}, \cdots, \sin_\theta^{i_n}\rangle$, $n>1, 0<i_w\leqslant L_\theta$, $w=1,2,\cdots,n$,满足以下两个条件:(1)$\sin_\theta^{i_n}\in G_\theta$;(2)$s_\theta^{i_w}(R_j)\geqslant s_\theta^{i_{w+1}}(R_j)$, $w=1,2,\cdots,n-1$, $R_j\in R_\theta$,其中 $o_j\geqslant \pi_\theta^{i_w}, \pi_\theta^{i_w}=\min\{o_k \mid s_\theta^{i_w}(R_k)>0,\ R_k\in R_\theta\}$,则称 P 为安全路径.

定义 2 安全状态　SD-RAS 中服务实例 θ,资源分配状态 $s_\theta^i\notin G_\theta, 0<i\leqslant L_\theta$,如果从 s_θ^i 出发存在一条安全路径,则称 s_θ^i 是安全状态.

定理 1　SD-RAS 中服务实例 θ,资源分配状态 $s_\theta^i\notin G_\theta, 0<i\leqslant L_\theta$,如果存在一个目标状态 $s_\theta^j\in G_\theta$,满足 $s_\theta^i(R_k)\geqslant s_\theta^j(R_k)$, $R_k\in R_\theta$ 且 $o_k\geqslant\pi_\theta^i$,则 s_θ^i 是安全状态.

本文假设 $o_1<o_2<\cdots<o_i<\cdots<o_M$,其中 o_i 是第 i 个资源 R_i 的优先级,应用根据优先级从高到低申请资源,因此,图 1 中 s_θ^3、s_θ^4、s_θ^6 和 s_θ^7 四个状态是安全状态.

4　最优 ODP³ 协议(OODP³)

本节介绍本文的主要贡献——OODP³ 协议,首先,我们介绍 OODP³ 协议的主算法,它是在 ODP³ 协议主算法的基础上做了两处改进而来;其次,我们重点阐述 OODP³ 协议的子算法——下一安全状态查找算法(NS³, Next Safe State Search),NS³ 算法建立在定理 2 的基础上,我们通过证明定理 2 来论证 NS^3 算法的正确性.

4.1　OODP³ 协议主算法

OODP³ 协议支持应用同时向多个物理站点发送并发资源请求,物理站点收到应用的请求后,为应用分配一定种类和数量的资源,如果应用的资源需求得不到满足,它将持有已分配的资源,并发送下一轮并发资源请求以申请剩余所需资源.OODP³ 协议从 SD-RAS 的初始状态出发找到一条到达某一目标状态的路径,并保证路径上每个中间状态都是安全状态来避免死锁和活锁.OODP³ 协议主算法的伪代码如图 2 所示.

算法 1　OODP³ 协议主算法

输入:目标状态集合 G_θ

输出:达到的目标状态 G_θ^l

描述:

1:　$Q_\theta\leftarrow\varnothing$;　/* 应用已持有的资源集合 Q_θ 初始化为空集 */

2:　while (TRUE) do

3:　　$T_\theta\leftarrow$ request $G_\theta^+ \setminus Q_\theta$;　/* 向资源提供者申请发送资源请求,资源请求量为 $G_\theta^+ \setminus Q_\theta$,实际申请到的资源为 T_θ */

4:　　for $l\leftarrow 1$ to N　/* 遍历所有目标状态 */

5:　　if ($G_\theta^l \setminus (Q_\theta\cup T_\theta)=\varnothing$) then　/* 判断本轮申请到的资源集合 T_θ 并上已持有的资源集合 Q_θ 是否满足某目标状态 */

6:　　cancel ($Q_\theta\cup T_\theta$) $\setminus G_\theta^l$;　/* 若目标状态 G_θ^l 已经满足,则释放多余资源 */

7:　　return l;　/* 返回该目标状态在 G_θ 中的标号 */

8:　　end if

9:　end for

10:　if (! safe($Q_\theta\cup T_\theta$)) then　/* 所有目标状态都未满足,判断 $Q_\theta\cup T_\theta$ 是否安全 */

11:　　$T_\theta'\leftarrow$ NS³(Q_θ, T_θ);　/* 若 $Q_\theta\cup T_\theta$ 不安全,调用 NS³ 子算法得到 T_θ 的子集 T_θ' */

12:　　cancel $T_\theta\setminus T_\theta'$;　/* 释放 T_θ 中的多余资源 */

13:　　$Q_\theta\leftarrow Q_\theta\cup T_\theta'$;　/* 将 T_θ' 分配给 θ,并更新 Q_θ 的值 */

14:　else

15:　　$Q_\theta\leftarrow Q_\theta\cup T_\theta$;　/* 若 $Q_\theta\cup T_\theta$ 安全,将 T_θ 分配给 θ,并更新 Q_θ 的值 */

16:　end if

17:end while

图 2　OODP³ 协议主算法的伪代码

OODP³ 协议用 Q_θ 表示应用已分配的资源集合,G_θ^+ 是综合考虑多个资源需求方案而形成的资源需求种类和数量,计算方式如式(1)所示:

$$G_\theta^+(R_i)=\mathrm{Max}_{i=1}^{n_\theta} G_\theta^i(R_i) \tag{1}$$

其中 $G_\theta=\{G_\theta^1, G_{\theta,\cdots}^2, G_\theta^{n_\theta}\}$,式(1)说明 G_θ^+ 中资源是需求方案集合 G_θ 中元素包含该类资源的最大值.运算符"\"的定义如式(2)所示:

$$S_1\setminus S_2=\begin{cases}\mathrm{Max}\{(S_1(R_i)-S_2(R_i)),0\}, & \text{if } R_i\in S_1\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

算法第 3 行:应用根据 G_θ^+ 和已分配资源集合向资源提供者发送并发资源请求,其中 $G_\theta^+ \setminus Q_\theta$ 表示所需资源与已分配资源的差值,即为目前尚且缺乏的资源种类和数量,资源提供者根据本地资源管理策略和可用资源数量为 θ 分配一定种类和数量的资源,保存于 T_θ 中.算法第 4 行到第 9 行,遍历目标状态集合,判断 T_θ 并上已分配资源集合 Q_θ 是否能够覆盖 θ 的某一目标状态,若是,则说明应用已分配到可满足某一目标状态的资源,OODP^3 主算法结束.否则,才转入算法第 10 行,该行根据定理 1 判断 $Q_\theta \cup T_\theta$ 是否是安全状态,若是,算法将 $Q_\theta \cup T_\theta$ 赋给 Q_θ,应用不释放 T_θ 中的任何资源,转到算法第 3 行,进行下一轮的资源申请,从 SD-RAS 上看,$Q_\theta \cup T_\theta$ 是一个安全的中间状态.如果 $Q_\theta \cup T_\theta$ 不是安全状态,算法第 11 行调用 NS^3 子算法,获得 T_θ 的子集 T_θ',使得 $Q_\theta \cup T_\theta'$ 是安全状态,且根据某目标函数,T_θ' 是所有子集中最优的,算法第 12、13 行释放 T_θ 中多余资源,并更新 Q_θ 的值,然后同样转到第 3 行,进行下一轮的资源申请.

尽管 OODP^3 主算法与 ODP^3 的过程相似,但是,我们已做了如下两点改进:(1)第 11 行,ODP^3 协议将寻找最优的 T_θ 子集问题归化为背包问题,对于规模较小的情况,ODP^3 采用穷举法,若规模较大,ODP^3 调用 KP 问题的启发式算法进行解决;而 OODP^3 通过 NS^3 子算法得到最优的 T_θ 子集,且 NS^3 子算法是多项式时间复杂度的;(2)OODP^3 将 ODP^3 的符号 $\uplus$ 改为并号 $\cup$,应用保留 Q_θ 和 T_θ 中各类资源之和,这更符合实际情况,比如,应用需要 $3R_1$ 资源,发送第一轮请求时获得 1 个 R_1,即 $Q_\theta(R_1)=1$,发送第二轮请求时得到 2 个 R_1,即 $T_\theta(R_1)=2$,则 $Q_\theta \uplus T_\theta(R_1)=2$,此时,应用显然应当保留 3 个 R_1,取 $Q_\theta(R_1) \cup T_\theta(R_1)=3$.

4.2 NS^3 子算法

NS^3 子算法的目标是查找下一个安全状态以最大限度地减小到达目标状态的时间,到达目标状态的时间可以指定一个目标函数来测量.当已分配资源集合 Q_θ 并本轮申请到资源集合 T_θ 不是安全状态时,OODP^3 释放 T_θ 中的一些资源(而 Q_θ 的资源从不被释放),使得 T_θ 的子集 T_θ' 并 Q_θ 是安全状态.为使应用尽快到达目标状态,应当尽量少地释放 T_θ 的资源.尽管目标函数可以有不同的定义方式,但是,目标函数的值是随着 T_θ 资源数量的减小而单调非递增的,下面给出目标函数的通用定义:

定义 3 目标函数:T_θ' 和 T_θ'' 为两个资源集合,当 T_θ' 所包含的资源数量小于 T_θ'' 所包含的资源数量时,若存在函数 $f(*)$,使 $f(T_\theta') \geqslant f(T_\theta'')$ 成立,则 $f(*)$ 可以作为衡量中间状态距离某目标状态远近的目标函数.

只要满足定义 3 的目标函数,NS^3 子算法都能找到 T_θ 的子集 T_θ',使得 $Q_\theta \cup T_\theta'$ 是安全状态,并且 $f(Q_\theta \cup T_\theta')$ 最小.ODP^3 协议定义中间状态和某目标状态之间的距离作为目标函数,距离的定义借鉴自 CPN (Colored Petri Net),如式(3)所示:

$$distance(Q_\theta') = \frac{\#\uplus_{i=1}^{n_\theta}(G_\theta^i \setminus Q_\theta'))}{n_\theta} \tag{3}$$

显然,式(3)所定义的距离符合定义 3.ODP^3 将查找 T_θ 的子集 T_θ',使得 $Q_\theta \cup T_\theta'$ 是安全状态、且 $f(Q_\theta \cup T_\theta')$最小这一问题归化为 KP 问题,KP 问题考虑了 T_θ 的所有子集,因而当资源的种类较多、数量较大时,穷举 T_θ 所有子集以得到最优解的算法是指数时间复杂度的.然而,本文认为无需考虑 T_θ 所有子集就可查找出 T_θ 的最优子集,我们首先提出定理 2,并进行详细的理论证明;然后基于定理 2,提出 NS^3 子算法.

定理 2 设 $S_\theta = Q_\theta \cup T_\theta$,$T_\theta'$ 是 T_θ 的一个子集,若 $S_\theta' = Q_\theta \cup T_\theta'$ 是安全状态,且 $f(S_\theta')$最小.由定义 2 可知,必存在一条安全路径从 S_θ' 到达某一状态,记为 G_θ^l,有 $G_\theta^l \in G_\theta$,设 G_θ^l 包含 m 种资源,记为:$R_{l1}, R_{l2}, \cdots, R_{lm}$.定义资源标号 $i \in [1, m]$满足如下条件:任取 $p \in [i+1, m]$,任取 $R_{lp} \in G_\theta^l$,都有 $S_\theta(R_{lp}) \geqslant G_\theta^l(R_{lp})$成立.为便于论述,定义集合 T_θ^*,满足:$T_\theta' = T_\theta - T_\theta^*$,那么,$T_\theta^*$ 的计算可分为以下两种情况:(1)当任取 $k \in [1, i]$,$Q_\theta(R_k)=0$ 时:$T_\theta^* = T_\theta(R_1)R_1 + T_\theta(R_2)R_2 + \cdots + T_\theta(R_i)R_i$;(2)当存在 $k \in [1, i]$,$Q_\theta(R_{lk}) \neq 0$ 时:$T_\theta^* = T_\theta$.

证明

(1)首先证明 S_θ' 是安全状态,按照计算 T_θ^* 的两种情况分开证明

(a)当任取 $k \in [1, i]$,$Q_\theta(R_k)=0$ 时,S_θ 中标号在 $[1, i]$区间内所有资源都来源于 T_θ.由于 $S_\theta = Q_\theta \cup T_\theta$ 和 $S_\theta' = Q_\theta \cup T_\theta'$,且 $T_\theta' = T_\theta - T_\theta^*$,所以 $S_\theta' = S_\theta - T_\theta^*$.又 $T_\theta^* = T_\theta(R_1)R_1 + T_\theta(R_2)R_2 + \cdots + T_\theta(R_i)R_i$,所以,任取 $k \in [1, i]$,$S_\theta'(R_k)=0$ 成立.由前述条件可知,任取 $p \in [i+1, m]$,任取 $R_{lp} \in G_\theta^l$,都有 $S_\theta(R_{lp}) \geqslant G_\theta^l(R_{lp})$成立,而任取 $k \in [i+1, m]$,都有 $S_\theta'(R_k) = S_\theta(R_k)$成立,所以 $S_\theta'(R_{lp}) \geqslant G_\theta^l(R_{lp})$也成立.所以,$S_\theta'$ 满足定理 1,S_θ' 是安全状态.

(b)当存在 $k \in [1, i]$,$Q_\theta(R_k) \neq 0$ 时,由于 OODP^3 协议主算法总是在安全状态上发出资源请求,因此,Q_θ 是安全状态,又 $T_\theta^* = T_\theta$,$S_\theta' = S_\theta - T_\theta^* = Q_\theta$,所以,$S_\theta'$ 是安全状态.

(2)其次证明存在安全路径到达 G_θ^l 的所有安全状

态中,S'_θ 是最优的.

(a)(反证法)当任取 $k\in[1,i]$,$Q_\theta(R_{lk})=0$ 时,假设存在一个比 S'_θ 更优的安全状态 S''_θ,且存在一条安全路径从 S''_θ 到达 G^l_θ.由定义 3,$f(S''_\theta)\geqslant f(S'_\theta)$,则 S''_θ 中的资源数量多于 S'_θ.如果设 $S''_\theta=S_\theta-T^{**}_\theta$,则 T^{**}_θ 的资源数量少于 T^*_θ.而 $T^*_\theta=T_\theta(R_1)R_1+T_\theta(R_2)R_2+\cdots+T_\theta(R_i)R_i$,若在 T^*_θ 中去掉任意一个资源得到 T^{**}_θ,该资源记为 R_j,所以,$T^{**}_\theta=T^*_\theta-R_j$.因为,$\pi^j_\theta=\min\{o_k\mid S''_\theta(R_k)>0,R_k\in R_\theta\}$,且必存在 $\pi^j_\theta\geqslant\pi^j_\theta$,且不满足任取 $R_{lp}\in G^l_\theta$,都有 $S_\theta(R_{lp})\geqslant G^l_\theta(R_{lp})$成立的条件,因此,$S''_\theta$ 不满足定理 1,S''_θ 不是安全状态,从而得出矛盾,得证.

(b)当存在 $k\in[1,i]$,$Q_\theta(R_{lk})\neq0$ 时,且 $S_\theta(R_{lk})\leqslant G^l_\theta(R_{lk})$,根据定理 1,如果存在一条安全路径从某个状态到达 G^l_θ,则必须丢弃 R_{lk},但是,OODP^3 主算法不允许丢弃 Q_θ 中的任何资源,因此,不可能存在新的安全状态,且存在一条安全路径从此状态到达 G^l_θ,因此,S'_θ 是最优安全状态,得证.

定理 2 证明了最优的下一安全状态必定是根据某目标状态构造而来的,构造方法实际上就是计算 T^*_θ 集合.基于定理 2,提出 NS^3 子算法,该算法的基本思想是:根据所有目标状态构造一个局部最优安全状态,局部最优安全状态是指由某目标状态所能构造出的最优安全状态,然后在所有局部最优安全状态中选择 $f(*)$ 最小的状态,该状态就为最优的下一安全状态,NS^3 子算法的伪代码如图 3 所示.

NS^3 子算法将由目标状态 G^l_θ 构造出的 T_θ 的子集记为 $T'_\theta[l](l\in[1,N])$,$f(T'_\theta[l])$最小的记为 $T'_\theta[k]$,OODP^3 将 $Q_\theta\cup T'_\theta[k]$作为下一最优安全状态.NS^3 子算法的第 2 行至第 5 行计算出定义 2 中 i 的值,第 6 行至第 10 行判断当 $j\in[1,i]$,是否存在 $Q_\theta(R_j)\neq0$,若存在,定理 2 的第(2)种情况满足,将由 G^l_θ 构造出的 T_θ 的子集 $T'_\theta[l]$赋为空值,并返回到第 1 行继续查看下一个目标状态,若不存在,则满足定理 2 的第(1)种情况,第 12 和 13 行对 $T^*_\theta[l]$和 $T'_\theta[l]$进行赋值.第 1 行至第 14 行的 for 循环结束之后,N 个目标状态所构造出的 T_θ 子集全部计算完毕,第 15 行取出 $f(T'_\theta[l])$最小的,返回给 OODP^3 主算法.

NS^3 子算法的时间复杂度为 $O(NM)$,是多项式函数.OODP^3 时间复杂度依赖于主算法中 while 循环的执行次数,while 循环的执行次数依赖于目标状态的资源数量,平均执行 $\mathrm{Avg}(|G^l_\theta|,l=1,2,\cdots,N)$次,因此,OODP^3 的时间复杂度为 $O(\mathrm{Avg}(|G^l_\theta|,l=1,2,\cdots,N)NM)$,也是多项式函数.

算法 2　最优下一安全状态查找子算法

输入:已分配资源集合 Q_θ;本轮申请到的资源集合 T_θ

输出:最优下一安全状态 $T'_\theta[k]$

描述:

```
1: for l ← 1 to N   /* 遍历所有的目标状态 */
2:   i ← M;
3:   while (Q_θ ∪ T_θ(R_i) ≥ G^l_θ(R_i))   /* while 循环用于计算定理 2 中 i 的值 */
4:   i ← i − 1;
5:   end while
6:   for j ← 1 to i   /* 判断 j∈[1,i],是否存在 Q_θ(R_j)≠0 */
7:     if (Q_θ(R_j) ! =0) then   /* 定理 2 的第(2)种情况满足 */
8:       T'_θ[l] ← ∅;   /* 由 G^l_θ 构造出的 T_θ 的子集 T'_θ[l]为空 */
9:       Go to step 1;   /* 回到第 1 步 */
10:      end if
11:  end for
12:    T*_θ[l] ← T_θ(R_1)R_1 + T_θ(R_2)R_2 + ⋯ + T_θ(R_i)R_i;   /* 定理 2 的第(1)种情况满足 */
13:    T'_θ[l] ← T_θ − T*_θ[l];
14: end for
15: k ← Min(f(T'_θ[l]), l = 1,2,⋯,N);   /* 在 N 个目标状态所构造出的 T_θ 子集中 */
16: return T'_θ[k];   /* 选择目标函数最小的,记为 T'_θ[k] */
```

图 3　NS^3 子算法的伪代码

5　仿真实验

本节通过仿真实验来评价本文所提出的 OODP^3 协议的性能,仿真程序使用 MATLAB 和 Java 编写,MATLAB 随机产生输入数据,输入数据是一组并发应用集合,每个应用包含多个目标状态和执行时间,Java 程序实现了 OODP^3、ODP^2 和 ODP^3 三种协议,读取由 MATLAB 产生的输入数据集,各应用根据上述几种协议的流程申请资源,应用的某目标状态得到满足时,应用开始执行,执行时间到达后就释放资源,直到并发应用集合内所有应用都执行完成,仿真程序终止.仿真实验中需要设置的参数列于表 1 之中.

表 1　实验参数

符号	描述
M	资源种类的数量
c	每种资源的数量
n	并发应用的数量
ET_i	第 i 个应用的执行时间
N_i	第 i 个应用的目标状态数量
CR	每个目标状态包含的资源种类数量
RT_i	第 i 个应用的退避时间

为了简化仿真实验,我们假设每种资源包含的资源数量相同(记为 c),每个目标状态包含的资源种类数量也相同(记为 CR),同时,假设第 i 个应用的执行时间 ET_i 为 10s 到 1000s 内的随机数.M、n 和 N_i 是影响资源

联合分配协议的主要因素,我们在不同实验场景中取不同的值.应用发送请求的退避时间指应用发送请求未能满足某一目标状态,而再次发送请求的时间间隔,为了在评估 OODP^3 协议性能时忽略退避时间产生的影响,我们将退避时间取最小值 1s(应用的执行时间是精确到秒的).

5.1 OODP^3 vs. ODP^3

第一组仿真实验比较 OODP^3 和 ODP^3 的性能,两种协议的区别在于计算 T_θ 子集的方法不同,如果 ODP^3 使用穷举法计算出 T_θ 的最优子集,OODP^3 和 ODP^3 将使得各个应用按照同样的路径达到目标状态,因此,OODP^3 和 ODP^3 引起的平均等待时间和资源利用率等方面的性能指标是完全相同的,它们的不同之处在于协议的执行速度,算法复杂度的分析已经能说明 OODP^3 速度是优于 ODP^3 的,本节通过仿真实验来比较 OODP^3 和 ODP^3 的执行时间,实验将 M 设为 10、c 设为 20、N_i 设为 3、CR 设为 5、RT_i 设为 1,n 从 5 增加到 50,n 每增加 5,随机产生 10 次输入数据,分别记录下 OODP^3 和 ODP^3 的执行时间,然后取 10 次的平均值作为结果,图 4 给出 OODP^3 和 ODP^3 的执行时间比较结果,仿真程序运行在 2.33 GHz Quad8200 处理器、3G 内存的普通 PC 上.从图 4 可以看出,OODP^3 的执行时间总是小于 ODP^3,这是因为 NS^3 子算法的速度比 0-1KP 问题穷举算法的速度快得多;但是,随着 n 的增大,OODP^3 和 ODP^3 的执行时间并非恒定增加,在很多情况下,应用的数量较大,应用的资源需求量却很小,应用的需求比较容易得到满足,查找下一个安全状态子算法就很少被调用,OODP^3 和 ODP^3 的执行时间较小、且基本接近.

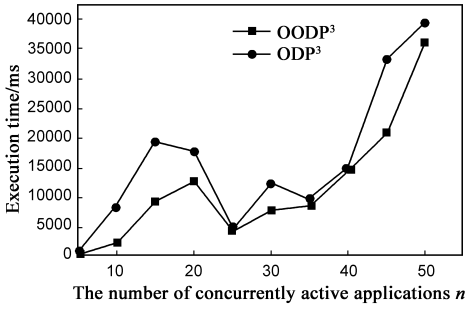


图4 OODP^{3}和 ODP^{3}的执行时间比较

5.2 OODP^3 vs. ODP^2

ODP^2 是早期流行的资源联合分配协议,它仅支持单目标状态,由于本文的仿真实验输入数据为每个应用定义了多个目标状态,因此,我们假设 ODP^2 随机挑选多个目标状态中的一个,应用从高到低依据优先级顺序申请资源,只有申请到的资源大于等于目标状态中资源数量时,才持有该资源,继续下一个资源的申请,直到满足目标状态为止.第二组仿真实验比较 OODP^3 和 ODP^2 的性能,我们首先定义如下两个性能指标:

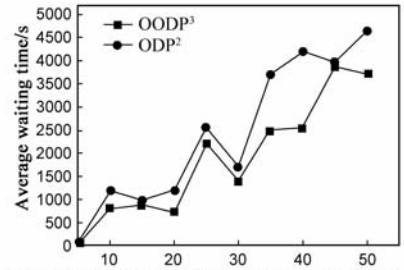
(1)平均等待时间:应用的等待时间是指发送第 1 个资源申请请求到它开始执行的时间,并发应用集合内所有应用等待时间的平均值就是平均等待时间.

(2)资源利用率:M 种资源类的所有资源被应用占用的时间所占的比例,可由式(4)计算,其中的 Makespan 指完成并发应用集合内所有应用花费的总时间.

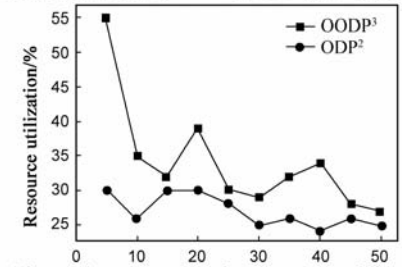
$$RU = \frac{\sum_{i=1}^{Mc} RT_i}{M \cdot c \cdot \text{Makespan}} \tag{4}$$

图 5 比较了 OODP^3 和 ODP^2 的平均等待时间和资源利用率随着并发应用集合数量 n 增加的变化,实验仍然将 M 设为 10、c 设为 20、N_i 设为 3、CR 设为 5、RT_i 设为 1,n 从 5 增加到 50,n 每增加 5,随机产生 10 次输入数据,分别记录下 OODP^3 和 ODP^2 的平均等待时间和资源利用率,然后取 10 次的平均值作为结果.OODP^3 的平均等待时间和资源利用率都比 ODP^2 优越,但是,有些情况下(比如图 5(a)中 n = 15 和 n = 45 的点),OODP^3 和 ODP^2 表现出接近的性能,这是由于应用需求的资源量相对于资源总量而言较小,应用容易申请到满足需求的资源组合,OODP^3 发送并发资源请求的优势体现的不明显;图 5 的结果还表明,OODP^3 协议的平均等待时间和资源利用率未必随着 n 的增大呈现单调上升或下降的趋势,其性能还受到具体资源需求量的影响,当应用的资源需求量相对于资源总量增大时,OODP^3 协议将比 ODP^2 显示出更好的性能优势.

图 6 比较了 OODP^3 和 ODP^2 的平均等待时间和资



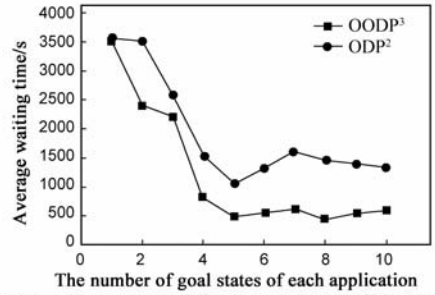
(a) 并发应用集合数量n变化对OODP^{3}和ODP^{2}的平均等待时间的影响



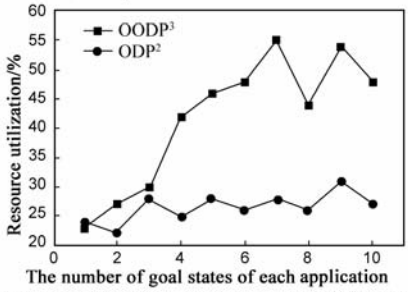
(b) 并发应用集合数量n变化对OODP^{3}和ODP^{2}的资源利用率的影响

图5 并发应用集合数量n变化对OODP^{3}和ODP^{2}的影响

源利用率随着目标状态数量 CR 增加的变化,实验仍然将 M 设为 10、c 设为 20、n 设为 25、CR 设为 5、RT_i 设为 1,N_i 从 1 增加到 10,N_i 每增加 1,随机产生 10 次输入数据,分别记录下 OODP^3 和 ODP^2 的平均等待时间和资源利用率,然后取 10 次的平均值作为结果.当应用只有一种目标状态时(即 $N_i=1$ 时),OODP^3 和 ODP^2 的平均等待时间几乎一样,ODP^2 的资源利用率甚至比 OODP^3 略高,这是由于 OODP^3 的安全状态与 ODP^2 持有资源、进行下一次申请的条件是一样的,两种协议使得应用遵循类似的资源申请过程.随着 N_i 的增加,OODP^3 的平均等待时间急剧下降、资源利用率大幅度增加,而 ODP^2 的两项性能指标无规律的变化,这是由于目标状态较多时,OODP^3 可以灵活地选择目标状态满足应用的需求,而 ODP^2 随机选择一种目标状态,不受到 N_i 增加的影响.



(*a*) 目标状态数量N_i变化对OODP^{3}和ODP^{2}的平均等待时间的影响



(*b*) 目标状态数量N_i变化对OODP^{3}和ODP^{2}的资源利用率的影响

图6　目标状态数量N_i变化对OODP^{3}和ODP^{2}的影响

6　OODP3 的进一步讨论

本文在评估 OODP^3 协议性能时忽略退避时间所产生的影响,我们将退避时间取最小值 1s,如此短的退避时间将导致应用和资源提供者之间的消息数量过大,而其中大量的请求并不能够申请到新的资源,因此,过短退避时间带来的很多申请资源请求是无意义的.反过来,退避时间过长将使得应用的等待时间变长、资源利用率变低.退避时间协议的研究是计算机网络领域的经典问题,Aloha、Ethernet 和 IEEE 802.11 协议等都定义了各自的退避时间协议[12,13],那么如何设计退避时间协议以减少消息传递的数量,却不影响平均等待时间和资源利用率呢?事实上,当一些应用处于持有部分资源、在等待剩余资源的状态时,才会根据退避时间协议发送新一轮的请求,这时一定有另一些应用正在执行,当没有应用执行完成而释放资源时,其他应用发送请求是不能申请到新资源的,即只有资源状态发生变化时,处于等待状态的应用发送新一轮的请求才可能申请到新资源.最优退避时间是从当前时刻到资源状态发生变化时刻的间隔,这需要资源提供者能获知资源状态变化的时间,并以应答消息的形式回馈给应用,应用根据资源提供者的应答消息确定其退避时间.

另外,资源提供者为资源申请请求分配资源的策略也是影响算法性能的因素之一,本文的仿真实验使用最简单的 FIFO 策略处理资源申请请求,对每个资源申请请求使用 best-effort 的策略,有关资源分配策略对 OODP^3 协议的影响将是我们下一步工作的重点.

7　结论

本文提出一种改进的死锁和活锁避免的分布式资源联合分配协议——OODP^3,OODP^3 仍然基于 ODP^3 的安全状态实现死锁和活锁避免,但是,OODP^3 的子算法 NS^3 能够保证在多项式时间复杂度内找到最优的下一安全状态,通过证明定理 2 论证了子算法 NS^3 的正确性;时间复杂度分析和仿真实验都表明 OODP^3 比 ODP^3 更加快速,而且 OODP^3 的平均等待时间和资源利用率等性能指标都比 ODP^2 优越,尤其在资源需求量大、目标状态数量多的情况下,OODP^3 的性能优势更加明显.在下一步的工作中,我们计划设计和实现基于 OODP^3 协议的分布式资源联合分配系统,对实际应用进行死锁和活锁避免的资源联合分配,从而在实际应用系统中评估 OODP^3 协议的性能.

参考文献

[1] I Foster, C Kesselman. 网格计算[M]. 金海,袁平鹏,石柯,译. 北京:电子工业出版社,2004.

[2] 陈康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009,20(5):1337 – 1348.
Cheng K, Zheng W M. Cloud computing: System instances and current research[J]. Journal of Software, 2009, 20(5): 1337 – 1448. (in Chinese)

[3] 刘志新,申妍燕,关新平. 一种基于 VCG 拍卖的分布式网络资源分配机制[J]. 电子学报,2010,38(8):1929 – 1934.
Liu Z X, Shen Y Y, Guan X P. A VCG auction based distributed mechanism for network resource allocation[J]. Acta Electronica Sinica, 2010, 38(8): 1929 – 1934. (in Chinese)

[4] R Buyya, D Abramson, S Venugopal. The grid economy[J]. Proc of the IEEE, 2005, 93(3): 698 – 714.

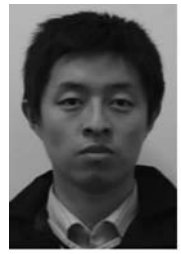
[5] A Abate, A. D′Innocenzo, M D D Benedetto, S Sastry. Understanding deadlock and livelock behaviors in hybrid control systems[J]. Nonlinear Analysis: Hybrid Systems, 2009, 3(2): 150

– 162.
[6] M Singhal. Deadlock detection in distributed systems[J]. IEEE Computer, 1989. 37 – 48.
[7] 廖名学,范植华. MPI 程序同步通信基本模型死锁检测[J]. 电子学报,2008,36(2):402 – 407.
Liao M X, Fan Z H. Deadlock detection in basic models of MPI synchronization communication programs[J]. Acta Electronica Sinica, 2008, 36(2): 402 – 407. (in Chinese)
[8] J Park. A deadlock and livelock free protocol for decentralized internet resource coallocation[J]. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2004, 34(1): 123 – 131.
[9] M Netto, R Buyya. Resource Co-allocation in Grid Computing Environments, Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications[M]. USA: IGI Global, 2010.
[10] J R Gonzalez de Mendivil, F Farina, J R Garitagoitia, C F Alastruey, J M Bernabeu-Auban. A distributed deadlock resolution algorithm for the AND model[J]. IEEE Trans Parallel and Distributed Systems, 1999, 10(5): 433 – 447.
[11] D Azougagh, J L Yu, J S Kim, S R Maeng. Resource co-allocation: A complementary technique that enhances performance in grid computing environment[A]. International Conference on Parallel and Distributed Systems (ICPADS 05)[C]. Los Alamitos, California: IEEE Computer Society, 2005. 36 – 42.
[12] D J Aldous. Ultimate instability of exponential back-off protocol for acknowledgment-based transmission control of random access communication channels[J]. IEEE Trans on Information Theory, 1987, 33(2): 219 – 223.
[13] F Calì, M Conti, E Gregori. IEEE 802.11 protocol: Design and performance evaluation of an adaptive backoff mechanism[J]. IEEE Journal on Selected Areas in Communications, 2000, 18(19): 1774 – 1786.

作者简介

伍之昂 男,1982 年 9 月生于江苏宜兴,博士,现为南京财经大学江苏省电子商务重点实验室讲师,主要研究领域为云计算,服务计算和数据挖掘.
E-mail: zawu@seu.edu.cn

曹 杰 男,1969 年 9 月生于江苏姜堰,博士,教授,主要研究领域为云计算、商业智能和数据挖掘.
E-mail: caojie690929@163.com



王有权 男,1984 年 10 月生于江苏徐州,博士生,主要研究领域为云计算和数据挖掘.
E-mail: youq.wang@gmail.com