

服务计算中避免死锁和活锁的资源分配算法

章 韵, 汤 楠

(南京邮电大学 计算机学院, 江苏 南京 210003)

摘 要: 服务计算系统资源层的目标是满足服务实例的资源分配需求, 保证其成功执行而不陷入死锁和活锁。首先将服务资源分配行为形式化为有限状态机; 其次提出一种避免死锁和活锁的资源分配算法, 它采用并发请求资源的方式, 且不需要并发服务实例之间交换消息。仿真实验结果表明, 该算法能够避免服务资源分配过程中的死锁和活锁, 表现出较高资源分配性能。

关键词: 服务计算; 资源分配; 死锁; 活锁

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2010)12-0069-05

Deadlock and Livelock Free Resource Allocation Algorithm in Service Oriented Computing

ZHANG Yun, TANG Nan

(College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: System resource layer in service oriented computing aims to allocate resources to various service instances to satisfy their requirements, so that each service instance can execute successfully and does not sink into both deadlock and livelock states. Firstly, the resource allocation behavior is formulated as a finite state machine. Secondly, a kind of deadlock and livelock free resource allocation algorithm is proposed, which adopts parallelization of requests for required resources and does not require communication between service instances. Experiment results show that the algorithm can effectively prevent the service resource allocation from deadlock and livelock, and exhibit good resource allocation performance.

Key words: service oriented computing; resource allocation; deadlock; livelock

1 引言

服务计算(Service Oriented Computing, SOC)是以服务为核心的新型计算模式。服务是自治、独立于平台的计算单元,可以在异构环境下针对不同的分布式应用进行快速、低成本的组合。服务计算基于面向服务架构(Service Oriented Architecture, SOA)^[1], SOA是一种追求敏捷性的面向服务的体系结构,它把业务逻辑和具体实现技术二者相分离,因而基于SOA所构建的应用系统具有灵活的底层基础设施和处理环境。

从功能上划分,服务计算模型可分为三个层次:应用层、服务实例层和系统资源层,分别主要解决以

下三方面的问题:业务流程聚合和管理,服务选择及服务资源管理^[2]。其中,服务资源管理处于模型的最底层——系统资源层,它作为服务实例执行的支撑系统,协同分配服务实例运行时所需的资源,保证服务实例的成功执行。

服务计算面向大规模网络环境,应用所提交的服务实例请求数量巨大,大量并发服务实例竞争有限服务资源的情况普遍存在,因此,在并发服务实例资源分配过程中极易导致死锁和活锁。当并发服务实例集中的每个服务实例占有集合中其它服务实例所要请求的资源,且各服务实例都不释放已占有的资源时,该组并发服务实例将处于永久阻塞状态,这种现象称为死锁。服务实例反复获得和释放同一资

源,却不能进一步获得其他资源,从而导致最终无法得到所有需要的资源,这种现象称为活锁.图1给出一个死锁和活锁的例子.已知服务资源 R_1, R_2, R_3 , 其数量分别为 2、3、2, 服务实例 $\theta_1, \theta_2, \theta_3$, 其对各类资源的需求量分别用元组表示为 $(2, 1, 1), (1, 2, 0), (0, 3, 2)$. $\theta_1, \theta_2, \theta_3$ 分别已占有 2 个 R_1 、2 个 R_2 和 2 个 R_3 , 为成功执行, 它们都还需要分配其他资源, 这就产生了死锁, $\theta_1, \theta_2, \theta_3$ 将无限期等待其他服务实例占用的资源. 如果服务实例 θ_1 获得资源 R_2 后又将其释放, 反复执行这样的操作而不能进一步获得 R_3 , 则将产生活锁.

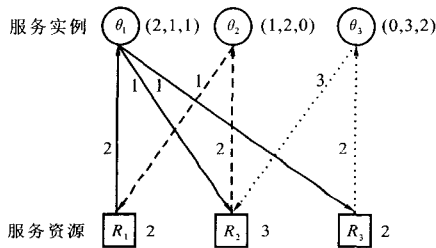


图1 死锁和活锁的例子

文献[3]提出了 SD-RAS(single-step distributed resource allocation systems)资源分配模型,并基于该模型提出一种采用死锁预防策略的算法—ODP³(order-based deadlock prevention protocol with parallel requests). SD-RAS 由并发应用集和有限资源集构成,不同的应用具有不同的目标资源分配状态,每个应用的目标资源分配状态也并不唯一.但是,ODP³ 算法在选择下一个安全状态时需要进子集选择,这是 NP 完全问题,文献[3]并未提出有效的算法.

文中提出一种死锁和活锁避免的资源分配算法,针对 ODP³ 算法在选择下一个安全状态上的不足,提出基于资源价值的新目标函数,将子集选择问题归化为 MKP 问题加以解决^[4].

2 服务资源分配模型

2.1 服务资源分配的形式化

文中借鉴 Park 提出的 SD-RAS 模型对服务资源分配问题进行建模.服务资源分配的 SD-RAS 模型由服务资源类型集合 $R = \{R_1, R_2, \dots, R_M\}$ 和服务实例集合 $I = \{\theta_1, \theta_2, \dots, \theta_K\}$ 构成,其中, $R_1, R_2, \dots, R_M$ 是数量(或容量)有限、功能不同的服务资源类型, $\theta_1, \theta_2, \dots, \theta_K$ 是不同的服务实例.

服务实例的成功执行一般需要分配一定数量的多种资源,因此服务实例要向对应的多个资源提供

者发送各自独立的资源请求. SD-RAS 模型定义了两个工作原语: 请求原语(request)和撤消原语(cancel). 服务实例使用请求原语向远程资源提供者请求分配一定数量的各种资源,使用撤消原语撤消已分配的资源. 文中假设一切消息通信可靠.

由于存在冗余或功能等价的资源,一个服务实例往往会有多个资源分配方案,服务实例满足其中一种方案即可成功执行,此刻所达到的资源分配状态即为目标状态,因此一个服务实例常常存在多个目标状态. 用 G_θ 表示服务实例 θ 所有目标状态的集合. 另外,一个资源分配状态如果真包含某目标状态中的所有资源,由于可以任意撤消冗余资源,则其也被视作目标状态. 服务实例 θ 刚激活时未分配任何资源,这种状态称为初始状态,记作 s_θ^0 . 从初始状态出发,服务实例 θ 通过发出系列资源请求或撤消原语向目标状态迁移.

文中利用有限状态机来形式化描述 SD-RAS 中服务实例的资源分配行为. 用 M_θ 表示服务实例 θ 资源分配行为的有限状态机模型,它描述了 θ 为到达目标状态可以采取的所有可能的资源获取序列. M_θ 的每个状态 $s_\theta^i, i = 0, 1, \dots, L_\theta$, 表示一个资源分配状态, L_θ 为 M_θ 中除初始状态外的状态总数. s_θ^i 可由该状态已分配的资源来定义. 例如, $s_\theta^i = R_1 + 2R_2$ 表示在该状态 θ 已获得一个 R_1 和两个 R_2 . $s_\theta^i(R_j)$ 表示在状态 s_θ^i 已分配的资源 R_j 数量, $R_j \in R_\theta$, 其中 R_θ 表示 G_θ 中全部目标状态所包含的资源类型集合. 图2中有两个目标状态,即 $G_\theta = \{R_1 + 2R_3, 2R_2\}$, 则 $R_\theta = \{R_1, R_2, R_3\}$. 由于 θ 在任何状态都可释放已分配的资源,故图2中指向非目标状态的迁移都是双向箭头. 因而该有限状态机中存在着环路,这意味着 θ 可能会陷入活锁.

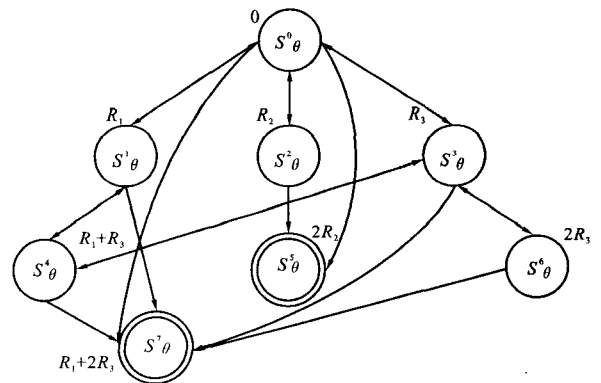


图2 服务资源分配有限状态机的例子

2.2 安全状态判定定理

死锁是由不当的资源分配顺序引起的,为预防

死锁,SD-RAS中表示不当资源分配顺序的中间状态是不允许的.因此,文中提出安全状态这一概念:如果服务实例从某一状态出发,能够不断获取更多资源,最终成功到达某一目标状态而过程中不陷入死锁,则称该状态是安全的.显然,服务实例 θ 的初始状态 s_θ^0 及 G_θ 中的各个目标状态都是安全的.后文将要介绍的死锁避免算法就是通过判断某一状态是否安全为服务实例提出可行的资源获取序列.只要资源获取序列所经历的中间状态都是安全的,算法将保证服务实例最终获得所有需要的资源.因此,死锁避免算法必然涉及资源分配状态的安全判定.文中安全状态的形式化定义如下:

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 $P = \langle s_\theta^{i_1}, s_\theta^{i_2}, \dots, s_\theta^{i_n} \rangle$, $n > 1, 0 < i_w \leq L_\theta, w = 1, 2, \dots, n$,满足以下两个条件:① $s_\theta^{i_n} \in G_\theta$;② $s_\theta^{i_w}(R_j) \geq s_\theta^{i_{w+1}}(R_j), w = 1, 2, \dots, n - 1, R_j \in R_\theta$.其中 $o_j \geq \pi_\theta^{i_w}, \pi_\theta^{i_w} = \min\{o_k \mid s_\theta^{i_w}(R_k) > 0, R_k \in R_\theta\}$,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 $s_\theta^i \notin G_\theta, 0 < i \leq L_\theta$,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.

定义1中的 o_j 表示资源 R_j 的次序.由安全状态的定义和服务资源分配行为的有限状态机模型,可以得到下面的安全状态判定定理,它是死锁避免资源分配算法的基础.

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

证明:考虑 M_θ 中一条路径 $\langle s_\theta^i, s_\theta^j \rangle$.在状态 s_θ^i 向相应的资源提供者发出一次并发资源请求后, θ 总有可能直接获得目标状态 s_θ^j 需要的所有资源,因此, M_θ 中一定存在有路径 $\langle s_\theta^i, s_\theta^j \rangle$.又已知 $s_\theta^i(R_k) \geq s_\theta^j(R_k)$,由定义1, $\langle s_\theta^i, s_\theta^j \rangle$ 是安全路径,再根据定义2即可得到结论.

定理1提供了一种计算效率较高的安全判定方法,它的安全判定只需要服务实例本身的局部信息,因此,计算复杂度不会随并发服务实例数的变化而变化.

3 死锁和活锁避免的资源分配算法

服务实例需要获得一定数量的多种资源才能成功完成,并往往存在多个资源需求目标,服务实例通

过调用死锁和活锁避免资源分配算法(Deadlock and Livelock Free Resource Allocation Algorithm, DLF-RAA)获取所需资源,以满足某一资源需求目标,因此,DLF-RAA算法以服务实例的多个资源需求目标为输入,以最终达到的某一资源需求目标作为输出.

为算法描述的方便,首先介绍两个运算符号: $\cup$ 和 $\setminus$.设资源分配状态 S 是由 $\{R_1, R_2, \dots, R_m\}$ 定义的多重集, $S(R_j)$ 表示 S 中 R_j 的数量, $j = 1, 2, \dots, m, \#(S)$ 表示 S 中各类资源的总数.若 $S = 2R_1 + 3R_2$,则 $S(R_1) = 2, S(R_2) = 3, \#(S) = 2 + 3 = 5$.已知由 $\{R_1, R_2, \dots, R_m\}$ 定义的状态 $S_1, S_2, S^+ = S_1 \cup S_2$ 和 $S^- = S_1 \setminus S_2$ 的定义如下:

$$S^+(R_j) = \begin{cases} \max\{S_1(R_j), S_2(R_j)\}, & R_j \in S_1 \text{ 或 } R_j \in S_2 \\ 0, & \text{其他} \end{cases} \quad (1)$$

$$S^-(R_j) = \begin{cases} \max\{S_1(R_j), S_2(R_j)\}, & R_j \in S_1 \\ 0, & \text{其他} \end{cases} \quad (2)$$

设 $S_1 = 2R_1 + 3R_2, S_2 = R_2 + 4R_3$,则 $S_1 \cup S_2 = 2R_1 + 3R_2 + 4R_3, S_1 \setminus S_2 = 2R_1 + 2R_2$.若 $S_2 \setminus S_1 = \emptyset$,则称 S_1 覆盖 S_2 .定义 $G_\theta^l = \cup_{i=1}^N G_\theta^i, G_\theta^l$ 表示 G_θ 中第 l 个目标状态,为 m 维向量,其第 j 维分量 $G_\theta^l[j]$ 表示 G_θ^l 中 R_j 资源的数量, N 为目标状态数.

3.1 DLF-RAA 算法

由于服务实例的资源分配过程在文中被形式化为有限状态机,其多个资源需求目标即是有限状态机的多个目标状态.DLF-RAA算法的目标就是从初始状态出发找到一条到达某一目标状态的路径,通过保证路径上每个状态的安全性来避免死锁和活锁.服务实例 θ 根据其多个资源需求目标向资源提供者发送并发资源请求,资源提供者根据本地资源管理策略和可用资源数量为 θ 分配一定量的资源.设 θ 通过此轮并发资源请求获得的资源集合为 T_θ ,如果 T_θ 并上已分配资源集合 Q_θ 能够覆盖 θ 的某一目标状态,DLF-RAA算法结束,并将该目标状态作为最终的资源分配方案返回.否则,DLF-RAA算法将对集合 $Q_\theta \cup T_\theta$ 进行安全性判断:如果 $Q_\theta \cup T_\theta$ 是安全的,DLF-RAA算法直接将该集合作为路径上的下一中间状态,并发送新一轮的并发资源请求;否则,DLF-RAA算法调用SSF(Safe Subset Find)子算法获得 $Q_\theta \cup T_\theta$ 的安全子集,并将该安全子集作为路径上的下一中间状态,然后再发送新一轮的并发资源请求.

DLF - RAA 算法的伪代码如下:

输入: 目标状态集 G_θ

输出: 作为资源分配方案的某一目标状态

描述:

```

1:  $Q_\theta \leftarrow \emptyset$ ;
2: while (TRUE) do
3:  $T_\theta \leftarrow \text{request } G_\theta^+ \setminus Q_\theta$ ; /* 并发资源请求, 将获得的
资源记于临时集合  $T_\theta$  中 */
4: if ( $G_\theta^+ \setminus (Q_\theta \cup T_\theta) = \emptyset$ ) then
5: cancel  $(Q_\theta \cup T_\theta) \setminus G_\theta^+$ ;
/* 如果满足某一目标状态  $G_\theta^+$ , 则释放冗余资源 */
6: return  $Q_\theta \cup T_\theta$ ; /* 算法结束, 返回 */
7: else
8: if (!safe( $Q_\theta \cup T_\theta$ )) then
9:  $T'_\theta \leftarrow \text{SSF}(Q_\theta, T_\theta)$ ;
/* 调用 SSF 子算法得到  $T_\theta$  的子集 */
10: cancel  $T_\theta \setminus T'_\theta$ ;
/* 释放  $T_\theta$  中未被选中的冗余资源 */
11:  $Q_\theta \leftarrow Q_\theta \cup T'_\theta$ ;
12: else
13:  $Q_\theta \leftarrow Q_\theta \cup T_\theta$ ;
/* 是安全状态, 则直接将  $Q_\theta \cup T_\theta$  作为下一状态 */
14: end if
15: end if
16: end while

```

DLF - RAA 算法通过释放一部分 T_θ 中的资源使下一状态安全, 而不释放已分配资源集合 Q_θ 中的资源, 也就是说, DLF - RAA 算法保证每个中间状态中各种资源的数量不会比其上一状态的少, 从而保证有限状态机是无回路的, 无回路有限状态机的资源分配过程是活锁避免的. DLF - RAA 算法又保证路径上的每个状态都是安全的, 这就确保资源按照优先级次序分配, 从而避免了死锁. DLF - RAA 是多项式时间算法, 其最坏时间复杂度为 $\Theta(\text{Max}_{l=1,2,\dots,N}(\#(G_\theta^l))Nn)$, $n = \#(T_\theta)$.

3.2 SSF 子算法

SSF 子算法的目标是从 T_θ 中选出一个子集 T'_θ , 使得 $Q_\theta \cup T'_\theta$ 是安全的. 为使 DLF - RAA 算法尽快到达目标状态, SSF 子算法需要释放尽量少的资源以获得 T_θ 尽量“大”的子集 T'_θ , 并且该算法优先释放低价值资源, 文中给出了资源价值的定义:

$$V_j = \frac{G_\theta^+[j]}{T_\theta[j] + Q_\theta[j]}, j = 1, 2, \dots, m \quad (3)$$

其中, G_θ^+ 、 T_θ 、 Q_θ 是 m 维向量, 第 j 维分量分别表示 G_θ^+ 、 T_θ 、 Q_θ 中所包含的第 j 类资源 R_j 的数量. V_j 反映了当前状态要达到目标状态缺少 R_j 资源的程度,

与 G_θ^+ 中资源相比, T_θ 和 Q_θ 中某类资源越多, 说明服务实例 θ 到达目标状态所需的这类资源量就越少, 因此, 服务实例 θ 可能并不急于获取这类资源, 这类资源的价值 V_j 就越小, 反之亦然.

文中将资源价值作为目标函数, SSF 子算法所解决的问题可重新表述为: 选出 T_θ 的真子集 T'_θ , 在满足 $Q_\theta \cup T'_\theta$ 安全性的前提下, 最大化 T'_θ 的资源价值. 在给出该问题的形式化定义之前, 需将定理 1 推广到推论 1: 服务实例 θ 的资源分配状态 S 是由 $R_\theta = \{R_1, R_2, \dots, R_m\}$ 定义的多重集, 现将 S 拆分为 $\#(S)$ 维向量表示的普通集合, $S = \{P_1, P_2, \dots, P_n\}$, $n = \#(S)$, $P_i \in R_\theta$. 定义 n 维向量 C 为 S 的权重向量, $C[i]$ 即为对应分量 P_i 的权重, 为 m 维布尔向量, 它表示了 P_i 的资源类型, 若 $P_i = R_j$, 则 $C[i] = (0, \dots, 1, \dots, 0)$, 其中, 第 j 位为 1, 即 $C[i][j] = 1$, 其余 $m-1$ 位都为 0. 设 G_θ 含有 3 种资源, 即 $m = 3$, $S = 2R_1 + 3R_3 = (2, 0, 3)$, $\#(S) = 5$, 可拆分成 5 维普通集表示为 $S = \{P_1, P_2, P_3, P_4, P_5\}$, 若 $P_1 = R_3$, 则 P_1 的权重 $C[1] = (0, 0, 1)$, 其中 $C[1][3] = 1$. 在上述向量表示的基础上, 得到定理 1 向量形式的推论.

推论 1 SD-RAS 中服务实例 θ , 目标状态集 G_θ 和状态 S , $S \notin G_\theta$. 若 $\exists G_\theta^+ \in G_\theta$, 满足 $\sum_{i=1}^n C[i][j] \geq G_\theta^+[j]$, $j = h, h+1, \dots, m$, $n = \#(S)$, 其中, $o_h = \min\{o_j \mid S(R_j) > 0, R_j \in R_\theta\}$, $h = 1, 2, \dots, m$, 则 S 为安全状态.

证明: S 可表示为 $\#(S)$ 维普通集, 所以 $\sum_{i=1}^n C[i][j]$ 表示 S 中资源 R_j 的数量. $\sum_{i=1}^n C[i][j] \geq G_\theta^+[j]$ 成立, $j = h, h+1, \dots, m$, 表示与目标状态 G_θ^+ 相比, S 中优先级次序较高的资源都已分配完毕, 根据定理 1, S 即为安全状态.

SSF 子算法所解决的问题可以归化为 MKP 问题, 将 T_θ 的 m 维向量形式拆分为 $\#(T_\theta)$ 维向量, $T_\theta = \{P_1, P_2, \dots, P_n\}$, $n = \#(T_\theta)$, n 维向量 C 为 T_θ 的权重向量, n 维向量 V 为 T_θ 的资源价值向量, 该问题可形式化为:

$$\text{Max} \sum_{i=1}^n V[i]x_i \quad (4)$$

满足: $\exists G_\theta^+ \in G_\theta$,

$$Q_\theta[j] + \sum_{i=1}^n x_i C[i][j] \geq G_\theta^+[j] \quad (5)$$

$$j = h, h+1, \dots, m$$

$$o_h = \min\{o_j \mid S(R_j) > 0, R_j \in R_\theta\}$$

$$S = Q_\theta \cup T'_\theta$$

$$x_i \in \{0,1\}, i=1,2,\dots,n, n=\#(T_\theta)$$

$x_i = 1$ 表示 P_i 被选出, 约束条件(5) 根据推论 1 保证 $Q_\theta \cup T'_\theta$ 的安全性. 上述形式化定义类似于 MKP 问题的定义, 但是与一般的 MKP 问题存在两点区别: (1) 约束条件是满足约束向量集合 G_θ 中的某一个; (2) 无需满足约束向量 G_θ 的全部 m 维约束, 而只需满足其中的 $m-h+1$ 维. MKP 是 NP 完全问题, 国内外众多学者对此问题进行了研究, 提出多种启发式算法^[5-7], 其中最简洁有效的是 Toyoda 提出的基于聚合资源方法的启发式算法^[5], 对 Toyoda 的算法稍加改进即可解决上述 MKP 问题的变例, 从而在多项式时间内找出安全子集.

4 仿真实验

本节通过与 ODP² 的比较说明 DLF-RAA 算法的性能. 实验中, 服务实例竞争的资源类型有 10 种, 各类资源的最大数量为 20; 每个服务实例最多需要 5 种资源, 它实际需要的资源种类数是 1~5 之间的随机值; 服务实例的运行时间为 10~1000s 之间的随机值; 每个服务实例最多有 5 种资源分配方案, ODP² 是随机选取其中一个分配方案作为目标.

通过观察服务实例到达率的变化对资源利用率和等待时间的影响来比较这两种算法. 设服务实例的到达时间间隔服从指数分布, 从 200s 到 10s 改变到达时间间隔, 在每个时间间隔的采样点处统计 100 个服务实例的实验结果.

定义两个性能指标: 资源利用率和等待时间.

$$RU = \frac{\sum_{i=1}^N \text{ReqNum}(i) \cdot \text{RunTime}(i)}{\text{TotalResNum} \cdot \text{Makespan}} \quad (6)$$

式(6)为资源利用率(RU)的定义: 资源用于执行 N 个服务实例所使用的负载占资源总负载的比例. 本实验中服务实例数 N 为 100. Makespan 是从第一个服务实例开始到达到最后一个服务实例执行完毕的时间段; TotalResNum 是资源总数量, 本实验中为 200. 服务实例运行时间(RunTime)是一个服务实例执行时占用资源的时间; ReqNum 是该服务实例需要的资源数. 等待时间是一个服务实例从到达开始执行之间的时间段.

图 3 给出了两种算法资源利用率的比较结果. 可以看出, DLF-RAA 算法的资源利用率始终比 ODP² 高. 随着服务实例到达率的增加, DLF-RAA 算法的资源利用率不断提高, 与 ODP² 的差距也在变大, 最终与之相比性能提高了大约 1/3.

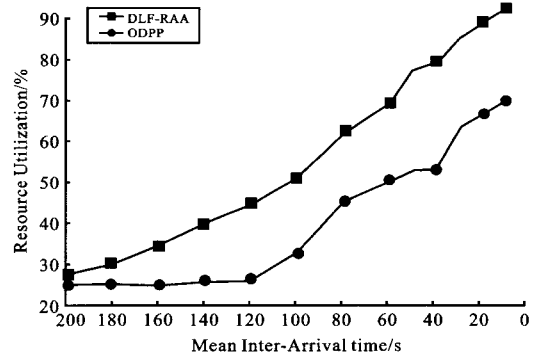


图3 DLF-RAA 和 ODP² (资源利用率)

图 4 给出了两种算法平均等待时间的比较结果. 平均等待时间就是 100 个服务实例等待时间的平均值. 从图中可以看出, 在到达率较小时, ODP² 和 DLF-RAA 算法的平均等待时间几乎没有差别, 这是因为服务实例此时可以很容易地获得空闲资源. 当到达率越来越大时, 服务实例之间开始产生资源竞争, ODP² 的等待时间越来越长, 尤其 100s 后呈爆炸式增长, 而 DLF-RAA 算法则无显著变化, 基本处于一个平稳状态.

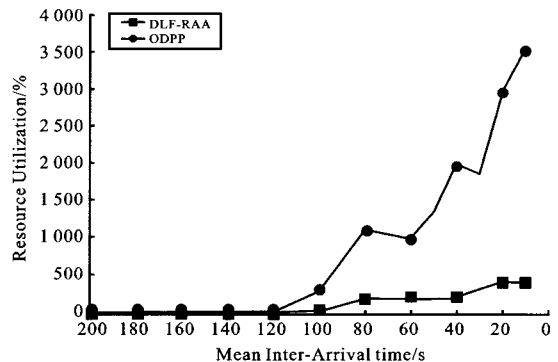


图4 DLF-RAA 和 ODP² (平均等待时间)

综上, 通过资源利用率和平均等待时间这两项性能指标的比较, DLF-RAA 算法的性能比 ODP² 高的多, 尤其是在服务实例高速到达的情况下, DLF-RAA 算法在资源利用率和平均等待时间上具有更明显的优势.

5 结束语

文中提出一种死锁和活锁避免的服务资源分配算法, 并将算法中选择下一安全状态的问题归化为 MKP 问题加以解决. 实验结果表明, DLF-RAA 算法可以有效避免死锁和活锁的同时能够获得较好的资源分配效率, 是比较理想的服务资源分配算法.

(下转第 77 页)

表1给出了对相同规模(遗传算法)性能评判的实验结论.其他智能算法也是基于概率的不确定性算法,同样方法可以写出信息熵的表达式.为更进一步论证智能算法的能耗特性,表2给出了以能量源为依据对遗传算法与其它算法(以模拟退火^[7-8]为例)不同规模算法的热力学性能差异的实验结论.

表1 遗传算法的热力学特性

进化代数	能量源	算法的不确定性	适应度	计算时间	算法的隐含并行性	信息量	信息熵
0	高	最大	低	短	最大	少	少
5	较高	较大	较低	较短	较大	较少	较少
10	较少	较小	较高	较长	较小	较多	较多
100	低	小	高	长	小	多	多

表2 不同规模算法的热力学特性

算法名称	能量源	算法的不确定性	计算时间	信息量	信息熵
遗传算法	高	较高	相同	较多	较多
模拟退火	小	高	相同	多	多

6 结束语

文中通过对热力学熵及信息熵进行分析和讨论,得出信息和能量之间存在的关系,并利用热力学中的耗散理论及算法的隐含并行性分析“熵”、“信息熵”、“信息量”、“能耗”之间关联.文中在评判智能算法的智能程度之外,拓展了对算法的能耗特性的研究,提出新的关注方向为热力学评判智能性,对智能优化算法的实现提出新的指标.

参考文献:

[1] Bennett C H. Logical reversibility of computation[J]. IBM Journal of Research and Development, 1973, 17 (6):525—532.
[2] 薛增泉. 热力学与统计物理[M]. 北京:北京大学出版

社,2004.
[3] 付朝江,张武. 非线性动力有限元重叠区域分裂的隐式并行算法[J]. 计算力学学报,2006,23(6):783—788.
[4] 李德毅,刘常昱,杜鹃,等. 不确定人工智能[J]. 软件学报,2004,15(11):1583—1594.
[5] Holland J H. Genetic algorithm and the optimal allocations of trials[J]. SIAM Journal of Computing, 1973 (2):88—105.
[6] 陈勇. 基于遗传算法的多值站内搜索设计与实现[J]. 微电子学与计算机,2008,25(11):219—221.
[7] Kirkpatrick S, Gelatt C D. Vecchi, optimization by simulated annealing[J]. Science, 1983,220(4598):671—680.
[8] 周鲜成,申群太. 基于微粒群和模拟退火的图像恢复研究[J]. 微电子学与计算机,2009,26(1):159—161.

作者简介:

郭本俊 男,(1975—),硕士研究生,讲师.研究方向为计算机软件与理论.
黄健 男,讲师.
陈东东 助理工程师.
王鹏 博士后.

(上接第73页)

参考文献:

[1] Huhns M N, Singh M P. Service-oriented computing: key concepts and principles[J]. Internet Computing, 2005, 9(1): 75—81.
[2] Papazoglou M P, Traverso P, Dustdar S, et al. Service-oriented computing: state of the art and research challenges[J]. IEEE Computer, 2007, 40(11): 38—45.
[3] Park Jonghun. 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.
[4] Park Jonghun. A scalable protocol for deadlock and live-lock free coallocation of resources in internet computing [C]// Proceedings of the 2003 Symposium on Applications and the Internet (SAINT' 03). USA: Orlando, 2003:66—73.
[5] Toyoda Yoshiaki. A simplified algorithm for obtaining approximate solution to zero—one programming problems[J]. Management Science, 1975, 21(12): 1417—1427.
[6] Loulou R, Michaelides E. New greedy-like heuristics for the multidimensional 0—1 knapsack problem[J]. Operations Research, 1979, 27(6): 1101—1114.
[7] 黄鲲,王知衍,翁小雄. 树形异构网络的启发任务调度算法[J]. 微电子学与计算机, 2007, 24(10): 134—136.

作者简介:
章韵 男,(1963—),博士研究生,副教授.研究方向为计算机技术在通信网络中的应用等.
汤楠 男,(1983—),硕士研究生.研究方向为服务计算.