

一种高效云任务调度博弈算法

段昌敏¹, 沈济南^{1,2}, 周慧华¹

(1 湖北民族学院 信息工程学院, 湖北 恩施 445000; 2 华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)

摘 要: 为了更加高效安全的进行云任务调度, 提出一种考虑资源可靠性的云任务调度博弈算法. 该算法将任务执行代价、资源利用代价、安全可信代价和负载均衡代价综合考虑到博弈效用函数中, 以最优博弈效用函数为目标, 将云任务调度问题形式化为非合作博弈模型进行求解. 算法求解过程划分为两个层次, 并分别利用 GA 算法和改进 MCT 算法对其进行求解, 最终得到非合作博弈的纳什均衡. 实验结果表明, 算法在保证任务完成时间和任务响应时间的同时, 可以降低任务执行的失效率, 并在不降低负载均衡度的情况下以较快的收敛速度整体提高任务执行效率.

关键词: 云计算; 任务调度; 博弈; 效用函数; 负载均衡

中图分类号: TP393.1

文献标识码: A

文章编号: 1000-7180(2017)03-0040-06

An Efficient Cloud Tasks Scheduling Game Algorithm

DUAN Chang-min¹, SHEN Ji-nan^{1,2}, ZHOU Hui-hua¹

(1 School of Information and Engineering, Hubei University for Nationalities, Enshi 445000, China;

2 School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: For implementing efficiently and safely cloud tasks scheduling, a task scheduling game algorithm considering resource reliability is presented in this paper. Tasks execution cost, resource utilization cost, security cost and load balance cost are considered comprehensively in the game's utility function in our algorithm. With optimizing the game's utility function as the goal, cloud tasks scheduling problem is formalized as a non-cooperative game model. This solving process is divided into two level. GA and improved MCT are used to solve the two level respectively, which eventually leads to get Nash equilibrium of non-cooperative game. Experimental results shows that our algorithm not only can guarantee task completion time and task response time, but can reduce the failure rate of task execution and improve the task execution efficiency integrally without reducing load balance degree and with a faster convergence.

Key words: cloud computing; task scheduling; game; utility function; load balance

1 引言

目前, 云任务调度已经取得了一些成果. 文献[1]利用改进的 DAG 图法进行延迟任务优化调度, 以确定最优的任务等待时间, 从而提高全局任务调度效率. 文献[2]提出一种基于改进离散粒子群算法的云任务调度算法, 算法采用自然数编码描述任务调度的粒子位置, 并以自适应惯性权重进行因子调

整, 以更少的迭代次数得到了任务调度最优解. 文献[3]采用了类似的方法, 以微粒子群算法进行任务调度优化, 但算法只优化了任务完成时间单一目标.

以上研究成果忽略了任务调度时对资源可信性的考虑, 使得调度系统在应对真实云环境任务调度时难以保障安全性. 文献[4]考虑了任务调度的安全因素, 以安全效益函数构建了一种安全驱动的任务调度模型, 但该模型只优化了任务调度成功率和系统吞吐

量,并没有考虑用户任务间调度的公平性.文献[5]基于社会学中的信任关系模型和贝叶斯认知模型,提出了基于信任关系的动态优先级云任务调度算法 Cloud-DLS,它可以有效降低用户任务调度的时间成本,在降低失效率方面确保了一定的安全性.文献[6]引入任务执行时间和用户信任度作为约束条件,提出一种调度算法在确保负载均衡的同时,一定程度上降低了执行时间.以上考虑安全因素的研究中,只考虑了用户任务本身的特点,优化目标通常只考虑了时间成本,而忽略了资源方执行任务的代价.

综合以上相关研究的不足,本文利用经济学中博弈论的方法,提出一种考虑资源可靠性和执行成功率的云任务调度算法,实现拥有更低失效率的云任务调度策略.

2 任务调度模型

令 n 表示任务数量,集合 $\text{Tasks} = \{1, 2, \dots, n\}$ 表示任务集, m 表示主机数,集合 $\text{Machines} = \{1, 2, \dots, m\}$ 表示主机集.基于预期执行时间 ETC 矩阵,对任务调度作如下形式化描述:

定义 1 负载向量 $WL = [wl_1, wl_2, \dots, wl_n]$, wl_i 表示任务 i 的计算负载,单位为 MI,即百万指令数.

定义 2 主机的计算能力向量 $CC = [cc_1, cc_2, \dots, cc_m]$,单位 MIPS,即每秒百万指令数.

定义 3 ETC 矩阵. $\text{ETC}[j][i]$ 定义为任务 j 在主机 i 上的执行时间估算.一般情况下,ETC 可由 WL 和 CC 向量计算得到:

$$\text{ETC}[j][i] = \frac{wl_j}{cc_i} \quad (1)$$

考虑资源可靠性与安全的云任务调度模型如图 1 所示.令 SD 为任务的安全需求, TL 为资源的可信度,则当 $TL \geq SD$ 时,表明可以执行安全的任务调度.由于并没有有效的方法评估资源的可信度和描述用户任务的安全需求参数,本文利用文献[7]提出的方法,将任务的 SD 和资源的 TL 定义在 $[0, 1]$ 范围内, SD 可视为认证请求、数据加密或访问控制等安全需求.令 P_f 表示主机失效的概率向量, $P_f[j][i]$ 为向量 P_f 中的一个元素,表示由于安全约束下任务 j 在主机 i 上执行时失效的概率.令该概率服从指数分布,表达式如等式(2)所示:

$$P[j][i] = \begin{cases} 1 - e^{-\alpha(sd_j - tl_i)}, & sd_j > tl_i \\ 0, & sd_j \leq tl_i \end{cases} \quad (2)$$

安全需求向量 SD 表示为 $sd_j (j \in \text{Tasks})$,主机

的可信度向量 TL 表示为 $tl_i (i \in \text{Machines})$, Tasks 为用户任务集, Machines 为主机集, $sd_j, tl_i \in [0, 1]$,其中,0 表示最低安全需求即最具风险的主机,1 表示最高安全需求即完全可信的主机,参数 α 表示失效因子,是作为模型全局参数的常量.若指数为负值表明失效率会随着 $sd-tl$ 值的增加而增加.

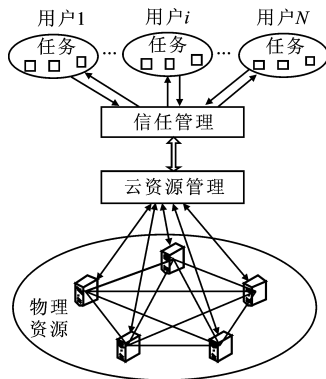


图1 安全保证的任务调度模型

3 非合作博弈任务调度

3.1 模型定义

云任务调度环境中,由于主机资源的使用权限对于所有用户是相同的,即用户行为是对称的,这表明用户之间在竞争资源时是一种非合作博弈行为.每个用户都尝试选择一种最优的策略进行任务与主机之间的映射,以最优优化其任务执行的效用函数.

非合作博弈论是研究具有完全理性的不同主体在策略相互依存情形下相互作用的理论,而非合作博弈资源分配正是侧重于从多个理性利益主体的行为所产生的相互影响与作用的分析,其中,个人的最优选择是其他竞争者选择的函数,通过寻求效用函数最优时的纳什均衡策略得到资源的优化配置方案.以下结合非合作博弈理论的定义要素,对云计算环境下的任务调度进行相关定义.

定义 4 (博弈参与者) 云计算环境中所有用户定义为博弈参与者(Game players),表示为 N .

定义 5 (用户策略) 任务调度方案 x 表示博弈者的策略空间(Strategy space),表示为向量 $x = [x^1, x^2, \dots, x^N]$,其中, $x^l = [x'_{k(l-1)+1}, x'_{k(l-1)+2}, \dots, x'_{k(l-1)+k_l}]$ 表示用户 l 的策略向量, k_l 为用户 l 的任务数,且 $k'_{(l-1)} = k_1 + \dots + k_{(l-1)}$,向量 x^l 表示用户 l 的决策变量,表明其任务分配的主机数.

定义 6 (效用) 用户的效用以效用函数(Utility function)集合表示,表示为 $\{U_i(\cdot)\}$, $\{U_i(\cdot)\} = \{U_1, U_2, \dots, U_N\}$,对于 $i \in \{1, 2, \dots, N\}$,

U_i 表示用户 i 的效用函数, $U_i(\cdot): x \rightarrow \mathbf{R}$.

定义 7 令参与博弈的用户数为 N , 一次调度的任务总量 n 可视为所有用户发送的任务总量, 即 $n = \sum_{i=1}^N k_i$, k_i 表示用户 i ($i = 1, 2, \dots, N$) 的任务量.

那么, 用户任务调度的非合作博弈可定义为一个三元组 $G_N, G_N = [N; \{J_i\}_{i=1,2,\dots,N}; \{Q_i\}_{i=1,2,\dots,N}]$, 其中, N 表示用户数量, $\{J_1, J_2, \dots, J_N\}$, $i = 1, 2, \dots, N$ 表示用户策略集, $\{Q_1, Q_2, \dots, Q_N\}$, $Q_i: J_1 \times J_2 \times \dots \times J_N \rightarrow \mathbf{R}$; $i = 1, 2, \dots, N$ 表示用户的效用函数集. 用户的策略向量 x^l 为策略空间 $J_l = J'_{k_{(l-1)}+1} \times \dots \times J'_{k_{(l-1)}+k_l}$ 的元素, 用户 l 参与博弈的效用 $U_l(\cdot)$ 定义为任务调度的代价, 表示为 Q_l . (注: 此时的效用 $U_l(\cdot)$ 定义为反向效用, 即代价 Q_l)

3.2 用户效用函数

博弈效用函数最优化过程即是最小化任务调度代价. 以下以代价函数形式定义用户的效用函数. 安全调度中, 除了要考虑用户的任务调度代价即任务执行代价外, 还需要考虑其他代价. 对于保证安全的任务调度, 用户需要支付额外的费用. 本文将所有代价整合为总代价函数 Q_l , $l \in \{1, 2, \dots, N\}$, 并将其定义为四个部分的和:

$$Q_l = Q_{l,e} + Q_{l,u} + Q_{l,s} + Q_{l,b} \quad (3)$$

式中,

$Q_{l,e}$ 表示用户任务的执行代价;

$Q_{l,u}$ 表示资源利用代价;

$Q_{l,s}$ 表示用户任务安全调度代价;

$Q_{l,b}$ 表示资源的负载均衡代价.

以下对这四个部分分别进行形式化定义:

(1) 用户任务执行代价

用户任务的执行代价定义为任务在主机上的平均完成时间. 令 $Machines(l)$ 表示分配给用户 l 的任务的主机集合, $Tasks(l)$ 表示用户 l 的任务集, 则执行代价可表示为:

$$Q_{l,e} = \frac{\sum_{j \in Tasks(l)} completion[j][i]}{completion_{m(l)} \times k_l} \quad (4)$$

式中, $completion[j][i]$ 表示任务 j 在主机 i 上的完成时间, 表示为式(5):

$$completion[j][i] = ETC[j][i] + ready[i] \quad (5)$$

式中, $ETC[j][i]$ 表示矩阵 ETC 中的元素, $ready[i]$ 表示预先分配到主机 i 上的任务的完成时间.

令 $completion_{m(l)}$ 表示用户任务的最大完成时间, 则

$$completion_{m(l)} = \max_{\substack{j \in tasks(l) \\ i \in Machines(l)}} completion[j][i] \quad (6)$$

(2) 资源利用代价

用户 l 的资源利用代价定义为执行其任务的主机的平均空闲时间所占比例:

$$Q_{l,u} = \sum_{i \in Machines(l)} (1 - \frac{Completion_{(l)}[i]}{Makespan(l)}) \times Idle_Factor[i] \quad (7)$$

式中, $Completion_{(l)}[i]$ 表示主机 i 的完成时间, $i \in Machines(l)$, 定义为:

$$Completion_{(l)}[i] = ready[i] + \sum_{\substack{j \in Tasks \\ x[j]=i}} ETC[j][i] \quad (8)$$

式中, $x[j]$ 表示给定的调度向量 x 中第 j 个分量的值, $Makespan(l)$ 表示 l 个用户的任务完成时间, 可以表示为对于所有的 $i \in Machines(l)$, $Completion_{(l)}[i]$ 的最大值.

$$(1 - \frac{Completion_{(l)}[i]}{Makespan(l)}) \times idle_Factor[i] \quad (9)$$

可视为对于给定的用户 l , 主机 i 的空闲时间, 该时间是主机 i 的空闲时间与分配到该主机上的用户 l 的所有任务的执行时间之比, 其中,

$$idle_Factor[i] = \frac{\sum_{j \in Tasks_{(l)}[i]} ETC[j][i]}{Completion_{(l)}[i]} \quad (10)$$

式中, $Tasks_{(l)}[i]$ 表示用户 l 分配到主机 i 的任务集.

(3) 安全调度代价

主机失效代价定义为失效率与任务在主机上的预期执行时间的乘积.

$$Q_{l,s} = \sum_{j \in Tasks(l)} \frac{P_f[j][i] \times ETC[j][i]}{(ETC)_{m(l)} \times k_l} \quad (11)$$

式中, $(ETC)_{m(l)}$ 表示用户 l 任务的最大预期执行时间, 即

$$(ETC)_{m(l)} = \max_{\substack{j \in tasks(l) \\ i \in Machines(l)}} ETC[j][i] \quad (12)$$

安全调度过程中, $Q_{l,s}$ 应该尽量最小化, 这意味着每个用户应尝试将任务分配到最可信的资源上, 即失效率 $P_f[j][i]$ 应该最小.

(4) 负载均衡代价

资源的负载均衡是云任务调度对于具有相同计算能力级别的资源应该在某种程度上尽可能多地利用可用资源, 而不是过于集中地使用某些资源. 主机资源在执行任务时的负载均衡代价为:

$$Q_{l,b} = \sqrt{\frac{\sum_{i=1}^m (\omega l_i - \bar{\omega l})}{m - 1}} \quad (13)$$

$Q_{l,b}$ 越小,表明主机资源在负载指标上负载越均衡,其中,

$$\overline{wl} = \frac{\sum_{i=1}^n wl_i}{n} \quad (14)$$

3.3 博弈过程求解

用户在博弈过程中的目标是最小化其代价函数 Q_l , 表示为 $\min Q_l$, $l = 1, 2, \dots, N$,

$$\min Q_l = \min_{x^l \in J_l} \{Q_l(x^1, \dots, x^N)\} \quad (15)$$

在每个博弈者在已知其他博弈者行为的情况下,博弈的均衡状态即为函数 $Q: J_1 \times J_2 \times \dots \times J_N \rightarrow \mathbf{R}$ 最小化的结果. 函数 Q 定义为:

$$Q(x^1, \dots, x^N) = \sum_{l=1}^N ([Q_l(x^1, \dots, x^N) - \min Q_l]) \quad (16)$$

用户的非合作博弈目标即是寻找这种均衡状态,即纳什均衡解.

为了求解函数 Q ,即需要最小化所有博弈者的代价函数. 笔者将博弈代价函数 Q 的最小化问题定义为两个层次. 层次 1: 求解函数 Q 最小化的全局问题; 层次 2: 求解用户代价函数 Q_l 最小化的局部问题.

为了定义最优优化机制,令 $x_{(0)}$ 为层次 1 中调度策略的起点,即 $x_{(0)} = [x_{(0)}^1, x_{(0)}^2, \dots, x_{(0)}^N]$, 其中 $x_{(0)}^l$ 表示调度策略 $x_{(0)}$ 的一个元素(即 $x_{(0)}$ 的一个子调度策略),由公式 x^l 定义的用户 l 的决策变量的值组成. 将 $x_{(0)}$ 复制 N 份并发送到层次 2 中,即每个用户得到一个初始调度策略,如图 2 所示. 每个用户通过改变自身任务的分配结果独立地优化其博弈代价函数. 作为最小化的结果,代价函数 Q 的最小值可通过式(17) 计算.

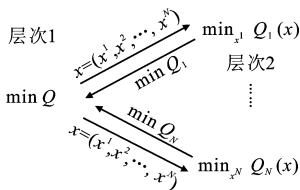


图 2 非合作博弈的效用最优优化求解

$$\begin{cases} \min Q_{1;(0)} = \min_{x^1 \in J_1} Q_1(x^1, x_{(0)}^2, \dots, x_{(0)}^N) \\ \dots \\ \min Q_{N;(0)} = \min_{x^N \in J_N} Q_N(x_{(0)}^1, \dots, x_{(0)}^{N-1}, x^N) \end{cases} \quad (17)$$

计算结果被送回层次 1, 博弈的目标函数 Q 通过调度策略 $x_{(0)}$ 进行计算. 同时, 向量 $x_{(0)}$ 实质上是 Q_l 最小化过程的起点.

为了求解两个层次中的调度问题,笔者在层次 1 中设计了一种基于遗传算法 GA 的调度方法进行

求解,在层次 2 中设计了一种基于 MCT (Minimization Completion Time) 的调度方法进行求解.

层次 1 的求解过程如下:

- (1) Input: Related parameters of GA
- (2) Output: Best individual population in scheduling
- (3) Generate the initial population P^0 of size s
- (4) Send the ready_times vectors of the machines corresponding to the individuals of the population P^0 to Layer 2
- (5) Receive the $\min Q_l$ values from Layer 2
- (6) Evaluate P^0
- (7) While not satisfy termination-condition do
- (8) Select the parameter pool T' of size λ : $T' = \text{Select}(P')$
- (9) Perform crossover procedure on pairs of individuals in T' with probability p_c , $P_{c,t} = \text{Cross}(T')$
- (10) Perform mutation procedure on individuals in $P_{c,t}$ with probability p_m , $P_{m,t} = \text{Mutate}(P_{c,t})$
- (11) Send the ready_times vectors of the machines corresponding to the individuals of the population $P_{m,t}$ to Layer 2
- (12) Receive the $\min Q_l$ values from Layer 2
- (13) Evaluate $P_{m,t}$
- (14) Create a new population P^{t+1} of size s from individuals in P' and/or $P_{m,t}$
- (15) Update iteration $t = t + 1$
- (16) End While
- (17) Return best found individual as solution

层次 2 的求解过程如下:

- (1) Input: Task set, Resource set, ETC vector, Safe demand, Trust level
- (2) Output: Optimal solution of tasks scheduling
- (3) Receive population schedules and ready_times of the machines from Layer 1
- (4) For all Schedule in the population
- (5) Compute the completion times of the machines in a given schedule
- (6) For all individual user
- (7) For all user's Task
- (8) Find the machine that gives minimum completion time
- (9) Allocate task to its best machine
- (10) Update the machine completion time
- (11) End For
- (12) Send the ready_times vectors to the individual game players
- (13) Compute the $\min Q_l$ in a given schedule for individual game palyers
- (14) Receive the $\min Q_l$ value from the individ-

ual game players

- (15) End For
- (16)Send the $\min Q_i$ values to Layer 1
- (17)End For
- (18)Return optimization solution of task scheduling

算法 2 基于最小完成时间算法 MCT,在 MCT 中,任务被分配到最早完成时间的主机上(定义为主机的就绪时间 ready_times 与该主机上执行所有任务的时间之和),并重复该过程直到没有任务待分配.在算法 2 中,笔者分别针对单个用户的任务应用 MCT 算法.首先,对所有来自层次 1 的调度方式进行遍历,并计算主机的完成时间.然后,MCT 算法在用户 1 的任务上进行运行,并重复这个过程到所有用户上.

4 仿真实验

为了评估本文的任务调度博弈算法,选取 CloudSim^[8]作为仿真实验模拟环境进行模拟实验,并将实验结果与经典任务调度算法 MIN_MIN 算法^[9]和 SDHGA 算法^[4]进行分析比较.选取的实验参数如表 1 所示.

选取任务执行时间跨度 Makespan、任务执行响应时间 Flowtime、任务执行失效率、资源平均利用率和负载均衡度作为性能评估指标.

从实验结果可以看出,任务量较少时,三种算法的性能差别并不大,这是由于此时可用主机资源相对充足,任务的资源竞争不明显.但随着任务量的增多,对有限资源的竞争加剧,本文的算法在任务完成时间和任务失效率方面就体现出了明显优势.

表 1 参数设置

参数类型	参数取值
任务总数	40,80,120,160,200
主机数	10
用户数	10
单个用户分配的任务数	20
任务到达率	0.8,单位:任务/秒/主机
任务安全需求	0.5~0.9,均匀分布
资源可性度	0.4~1.0,均匀分布
失效因子	2
种群规模	50
交叉概率	0.7
变异概率	0.2
迭代次数	50

如图 3 和图 4 所示,在资源平均利用率和资源负载均衡度方面,由于考虑了任务调度的负载均衡

代价,本文算法的负载均衡度明显高于另外两种算法,MIN_MIN 算法由于优先选择执行速度快的主机,负载均衡度量最低. SDHGA 为了追求高吞吐量,一定程度地提高了负载均衡度.

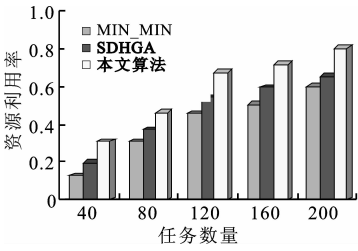


图 3 资源平均利用率

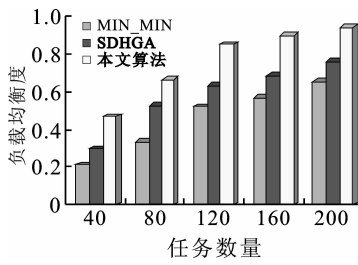


图 4 负载均衡度

基于类似的原因,在资源平均利用率方面,三种方法也表现出了相同的趋势.但是从图中可以看出,初始阶段,资源利用率会随着任务量的增加迅速地增加,随后增加趋势有所放缓,但趋势不变,这是由于资源占用达到一定饱和状态,任务在等待执行的原因造成的.

图 5 通过是否产生稳定不变的代价观察本文提出算法的收敛性.图中显示了不同规模的任务量下代价函数值随算法迭代次数的变化.可以看出,随着调度任务量的增加,博弈的代价是递增的.初始情况下,代价是较低的,主要是由于此时还没有执行较多任务.随着算法的运行,本文算法的收敛性是明显的,说明算法能较快找到非合作博弈的纳什均衡解,大约在迭代至 15 次左右时,算法就可以达到完全收敛,此时,没有任意一个用户会脱离此时所采取的调度策略,说明算法具有较好的鲁棒性.

综上,由于可用主机资源的局限性,为了更好满足调度任务的时间要求和安全调度要求,本文的调度算法总是在尽可能将任务调度到最合适的资源上进行任务调度,不仅拥有较小的单个任务完成时和任务完成总时间,而且还能一定保证资源的可性度,保证负载均衡的情况下极大降低任务调度的失效率,尤其在任务量增加时,算法的优势更加明显,并能较快的达到收敛状态,得到纳什均衡解.

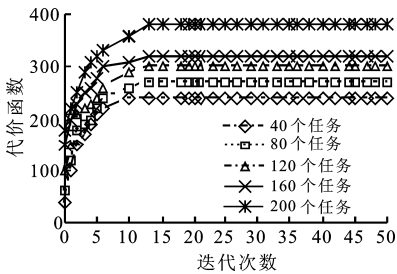


图 5 本文算法收敛性

5 结束语

本文提出了一种满足安全需求的云任务调度算法,算法结合非合作博弈论,将云任务调度问题形式化为用户的非合作博弈,博弈目标是最小化综合考虑了任务执行代价、资源利用代价和资源可信代价的博弈效用函数。结果表明,算法在保证任务完成时间和任务响应时间的同时,还可以提高任务执行的可靠性,为云任务的执行与资源分配提供较好的安全保障。

参考文献:

[1] 杨照峰,王启明,吕海莲. 基于任务延迟的云计算资源调度算法研究[J]. 计算机测量与控制, 2014, 22(2): 499-502.

[2] 沈恺涛,胡德敏. 基于云计算和改进离散粒子群的任务调度研究[J]. 计算机测量与控制, 2012, 20(11): 3070-3072.

[3] 胡德敏,户静,余星. 云计算环境下基于微粒群的虚拟机任务调度算法[J]. 计算机测量与控制, 2014, 22(4): 1189-1192.

[4] 朱海,王宇平. 安全驱动的实时任务调度遗传算法[J]. 系统工程与电子技术, 2010, 32(4): 854-859.

[5] Wang W, Zeng G, Tang D, et al. Cloud-DLS : dynamic trusted scheduling for cloud computing[J]. Expert systems with applications, 2012, 39(3): 2321-2329.

[6] Randies M, Lamb D, Taleb A. A comparative study into distributed load balancing algorithm for cloud computing[C]// Proceedings of the IEEE 24th international conference on advanced information networking and applications Workshop. Perth, Western Australia, 2010: 551-556.

[7] Song S, Hwang K, Kwok Y. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling[J]. IEEE Transactions on Computers, 2006, 55(6): 703-719.

[8] Rodrigo C, Rajiv R, Anton B, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software: practice and experience, 2011, 41(1): 23-50.

[9] He S, Sun H, Laszewski G. QoS guided Min-Min heuristic for grid task scheduling[J]. Journal of Computer Science and Technology, 2003, 18(4): 442-451.

作者简介:

段昌敏 男, (1973-), 硕士, 副教授. 研究方向为云计算和信息安全. E-mail: 2018371612@qq.com.

沈济南 男, (1980-), 硕士, 副教授. 研究方向为云计算与云安全、访问控制.

周慧华 女, (1973-), 博士, 教授. 研究方向为信息安全.