一种可靠性驱动的云工作流调度遗传算法*

魏秀然[1]，王 峰[2]
(1. 河南农业大学 信息与管理科学学院，郑州 450046；2. 华北水利水电大学 软件学院，郑州 450045)

摘 要：为了解决云环境中工作流调度的可靠性问题，提出了一种基于可靠性驱动信誉度模型的工作流调度遗传算法 RDR-GA。算法以工作流执行跨度 makespan 与可靠性最优化为目标，设计了一种基于时间依赖的可靠性驱动信誉度模型，通过该模型可以有效评估资源可靠性。同时，为了寻找遗传最优解，算法设计了新的遗传进化和评估机制，包括：以进化算子对调度解中的任务—资源映射进行遗传进化；以两阶段 MAX-MIN 策略评估并决定调度解的任务执行序列。仿真实验结果表明，满足可靠性驱动的信誉度算法不仅能够以更精确的信誉度改善工作流应用执行可靠性，而且能够以比同类遗传算法更快的收敛速度得到进化更优解。

关键词：云计算；工作流调度；遗传算法；信誉模型
中图分类号：TP393；TP301.6 **文献标志码**：A **文章编号**：1001-3695(2018)05-1390-05
doi：10.3969/j.issn.1001-3695.2018.05.023

Cloud workflow scheduling genetic algorithm satisfying reliability driven

Wei Xiuran[1], Wang Feng[2]
(1. *College of Information & Management Science, Henan Agricultural University, Zhengzhou* 450046, *China*; 2. *College of Software, North China University of Water Resources & Electric Power, Zhengzhou* 450045, *China*)

Abstract: In order to solve the reliability problem of workflow scheduling in cloud computing environment, this paper proposed a workflow scheduling genetic algorithm based on reliability-driven reputation model. This algorithm defined the makespan and reliability of workflow scheduling as the optimization objectives, and designed a reliability-driven reputation model based on time dependent. It could effectively evaluate the reliability of a resource by this model. At the same time, to find genetic optimal solution, this algorithm designed a novel evoluation and evaluation mechanism, included: the evolution operators evolved the task-resource mapping of a scheduling solution and the evaluation step determined the task order of solutions by using the two-phase MAX-MIN strategy. The simulation experimental results show that the satisfying reliability-driven reputation algorithm not only can improve the reliability of an workflow application execution with more accurate reputations, but also can provide and evolve to better solutions with a faster convergence speed than another genetic algorithm.

Key words: cloud computing; workflow scheduling; genetic algorithm; reputation model

0 引言

云计算作为一种新兴的计算模式[1]，正在受到越来越广泛的关注。它可以将分布于互联网上的计算、存储及各类 IT 资源进行有效整合，并利用抽象化及虚拟机的多层次实现技术，以按需与即付即用的提供方式有效提供给外部用户使用。云计算资源调度的目标是实现各类任务与云资源之间在满足有效时间和空间的同时得到供求双方满意的映射关系[2]。尤其在利用云计算解决科学工作流调度问题时，由于工作流任务间的相互影响与相互依赖关系，其调度技术更将直接影响任务执行成功率[3]。

云计算环境中的科学工作流应用通常表现为海量数据密集型任务处理，主要特点表现为[4]：a) 工作流任务提交与计算过程中动态多变，通常为有向无环图结构；b) 对资源可靠性要求更高，需要设置有效的容错机制。综合以上两点考虑，在大规模云系统中，工作流调度除了需要考虑任务执行跨度 makespan 问题，更必须考虑可靠性问题。同时，为了实现可靠性工作流调度，需要重点解决两个问题，即如何评估单个资源的可靠性和如何基于资源可靠性信息寻找可靠调度方案。

相关研究中，文献[5]通过赋予任务不同优先级，设计了一种异构最快完成时间调度算法 HEFT，从而实现了任务调度时间最小化。文献[6]对 HEFT 算法进行了扩展，提出了一种基于预算约束的异构最快完成时间算法 BHEFT，充分考虑了任务调度时的最优预算约束问题。文献[7]提出一种异构预算约束调度算法 HBCS，通过定义代价因子调整可用预算与最低廉价格可能性的比例，实现调度优化。HEFT、BHEFT 和 HBCS 均是以执行跨度 makespan 为单目标进行最优化，忽略了调度可靠性问题。文献[8]在考虑任务优先序列并结合复制的思想，提出一种基于可靠性的工作流调度算法 CRB，算法可以有效实现执行时间与执行失败率的同步降低，提高整体可靠性。文献[9]同步考虑工作流的安全需求问题，在满足用户时间和成本约束的基础上，设计了一种安全约束算法 VNPSO，并利用变近邻 PSO 对工作流调度最优化问题进行了求解，实现了寻优能力较强的可靠性调度。遗传算法 GA 是求解工作流

收稿日期：2017-02-15；**修回日期**：2017-03-27 **基金项目**：河南省重点科技攻关项目(152102210112)；河南省教育厅科学技术研究重点项目(13A520713)

作者简介：魏秀然(1975-)，女，河南郑州人，实验师，硕士，主要研究方向为软件技术、人工智能及其应用(hbwxr9416@163.com)；王峰(1970-)，男，河南安阳人，副教授，硕士，主要研究方向为图像处理、软件技术.

调度多目标优化的另一种有效方法。文献[10]提出的双目标调度算法 BGA 同样是以执行跨度 makespan 和可靠性作为优化目标,但算法执行过程中会违背任务依赖性并可能产生无效解,同时,传统的遗传算法对调度解的进化均是随机进行的,这会导致遗传寻优收敛速度较慢。综合以上工作,考虑可靠性的工作流调度研究的问题在于:a)从资源角度来看,其应用的信誉度模型仅仅依据完成任务的成功率评估资源信誉度,而忽略了任务运行时(任务大小)的影响;b)从任务角度来看,已有的信誉度模型基于资源信誉度为单个资源上的所有任务分配相同可靠性(成功率),忽略了任务在不可靠资源上执行时间越长,其成功率就越低的情况。

为了解决工作流调度可靠性问题,本文从信誉度的角度设计了一种新的工作流调度遗传算法。为了评估资源可靠性,算法通过利用任务失效率,考虑任务运行时特点的方式定义资源信誉度,并应用指数失效模型评估资源实时信誉度,从而实现任务的可靠性调度。

1　系统模型

表 1 给出了本文使用的符号及其含义说明。以有向无循环图 DAG 建立工作流调度模型,表示为 $G=(V,E)$,其中,V 表示节点 $v_i(1\leqslant i\leqslant n)$ 集合,代表工作流应用的任务,E 表示边 $e(i,j)(1\leqslant i\leqslant j\leqslant n)$ 集合,代表任务 v_i 和 v_j 间的依赖关系,v_i 为父任务,v_j 为子任务。没有父任务的任务称为入口任务,没有子任务的任务称为出口任务。对于任务 v_i,其权重 $|v_i|$ 表示该任务执行的指令数(任务大小)。本文重点关注于计算密集型应用工作流的调度问题,因此未建立任务间的通信时间模型。

表 1　符号说明

符号	参数含义
v_i	DAG 中的一个任务节点
$e(i,j)$	任务 v_i 与 v_j 间的依赖关系
$\|v_i\|$	任务 v_i 的指令数量
r_i	云系统中的单个资源
γ_i	资源 r_i 的单位指令执行时间
rdr_i	资源 r_i 的可靠性驱动的信誉度
$M(i)$	任务 v_i 调度的目标资源
s_i	资源 r_i 的一个间隔时间的开始时间
e_i	资源 r_i 的一个间隔时间的结束时间
rt_i	在当前时间间隔中资源 r_i 的总 CPU 时间
c_i	在当前时间间隔中资源 r_i 的任务失效数
rdr_i^{ti}	时间间隔 t_i 中资源 r_i 的可靠性驱动信誉值
t_i^{avail}	任务 v_i 的可用开始时间
idle(r_j)	资源 r_j 的空闲时间
t_i^s	任务 v_i 的开始时间
t_i^e	任务 v_i 的结束时间
t_S^j	资源 r_j 在调度 S 中完成其所有任务的时间
R_S	调度 S 中应用的成功率
fail(S)	调度 S 的失效因子
time(S)	调度 S 的执行跨度 makespan
imprt(i)	DAG 中以任务 v_i 为起点的最长路径长度
$p(i)$	任务 v_i 的优先级

令 $R=\{r_1,r_2,\cdots,r_m\}$ 表示云系统中 m 个可用资源,每个资源 r_i 具有两个特征属性:a)γ_i 表示以单位指令执行时间表示的资源 r_i 的计算速度,即执行单个指令的时间;b)rdr_i 表示资源 r_i 的可靠性驱动的信誉度,信誉度 rdr_i 描述资源 r_i 的失效率。令 $M:V\rightarrow R$ 表示映射函数,因此,$M(i)=r_j$ 表示任务 v_i 调度至资源 r_j。

2　信誉度计算模型

云计算环境中,许多离散事件均可能导致应用任务的失效,如请求服务的不可用、资源超载或恶意攻击等,以上事件均是独立且随机发生的。因此,笔者提出使用指数分布模型建立资源失效模型。失效密度函数表示为 $f(t)=\lambda e^{-\lambda t}(t\geqslant 0)$,其中,$\lambda$ 表示资源的失效率。令 num_fails 表示工作流执行周期为 run_time 时的单个资源失效次数。那么,可以计算出失效率 λ 为失效平均时间 MTTF(mean time to failure)的倒数:

$$\lambda=\frac{1}{\int_0^{\infty}\lambda x e^{-\lambda x}\mathrm{d}x}=\frac{1}{\text{MTTF}}=\frac{\text{num\_fails}}{\text{run\_time}} \tag{1}$$

由于传统的信誉度度量方式无法直接预测资源的失效率,且没有考虑执行时间对失效率的影响,笔者定义了一种时间依赖的信誉度模型,称为可靠性驱动信誉度模型。

定义 1　对于资源 r_i 的可靠性驱动信誉度 rdr_i 可理解为单位时间内任务失效的概率,即资源无法完成所调度工作流任务的概率。

算法 1 给出了连续时间间隔内(时间窗口大小为 T_{window})可靠性驱动信誉度的计算方法。在每个时间间隔内,系统为每个资源 r_i 维持一个信誉度向量 $\text{repu}_i=(s_i,e_i,rt_i,c_i)$,其中:$s_i$ 表示时间间隔的开始时间;e_i 表示时间间隔的结束时间;rt_i 表示在当前时间间隔中资源 r_i 的总 CPU 时间;c_i 表示在当前时间间隔中资源 r_i 的任务失效数。

算法 1　可靠性驱动信誉度计算方法

```
1  foreach resource r_i do
2    rdr_i = rdr_0 i = rdr^initial
3    t_i←1
4    s_i = e_i = current_time
5    rt_i←0
6    c_i←0
7  while there is a reputation record testimony_j^i do
8    if e_j^i < s_i + T_window then
9      //current interval
10       c_i←c_i + r_j^i
11       rt_i←rt_i + (e_j^i − s_j^i)
12       e_i←max(e_j^i, e_i)
13       remove record testimony_j^i
14       compute rdr_i using equation (2)
15     else
16       //next interval
17       rdrt_j^i←rdr_i
18       t_i←t_i + 1
19       s_i = e_i = s_i + T_window
20       rt_i←0
21       c_i←0
```

算法说明:算法在行 1 ~ 6 中对每个资源 r_i 的第一个时间间隔的信誉度向量 repu_i 进行初始化,在调度至资源 r_i 的任务 v_j 成功完成或失效后,算法给出信誉度记录 $\text{testimony}_j^i=(s_j^i,e_j^i,c_j^i)$,其中,$s_j^i$ 和 e_j^i 分别表示任务 v_j 的开始时间和结束时间,c_j^i 表示该任务执行期间的失效次数。如果任务失效,设置 $c_j^i=1$,否则 $c_j^i=0$。然后,算法利用该记录更新资源 r_i 的信誉度向量 repu_i,如行 10 ~ 12 所示。

信誉度向量 repu_i 更新之后,可以计算资源 r_i 的实时失效

率,此时,当前时间间隔的长度为 $e_i - s_i$。在当前间隔中资源 r_i 所贡献的任务执行时间 rt_i 中,资源 r_i 有 c_i 次任务失效。而在当前间隔的剩余时间 $e_i - s_i - rt_i$ 中,资源 r_i 假设使用了在前一个时间间隔 $t_i - 1$ 中观察到的信誉度向量,因此,在当前间隔的剩余时间中任务失效数为 $rdr_{i(e_i - s_i - rt_i)}^{t_i - 1}$,其中,$rdr_i^{t_i - 1}$ 表示在前一个时间间隔 $t_i - 1$ 中记录的资源 r_i 的信誉度,此时,可以通过等式(2)推导出资源 r_i 的实时信誉度 rdr_i,而 $(e_i - s_i - rt_i)/(e_i - s_i)$ 可理解为前一时间间隔中资源 r_i 的信誉度时间衰减因子。

$$rdr_i = \frac{c_i + rdr_i^{t_i - 1}(e_i - s_i - rt_i)}{e_i - s_i} = \frac{rt_i}{e_i - s_i} \times \frac{c_i}{rt_i} + \frac{e_i - s_i - rt_i}{e_i - s_i} \times rdr_i^{t_i - 1} \tag{2}$$

在当前时间间隔 t_i 结束时,资源 r_i 的实时信誉度 rdr_i 记录为 $rdr_i^{t_i}$,如行 17、18。然后,算法在下一个时间间隔 $t_i + 1$ 中开始于另一个信誉度向量,如行 19 ~ 21。对于初始时间间隔,假设每个资源 r_i 的可靠性驱动信誉度 rdr_i^0 为 rdr^{initial},如行 2 所示,rdr^{initial} 为所有资源的初始信誉度,可设置为相对较高的失效率,通过这种方式,可以激励资源提供方提供更高质量的服务以改善其信誉度。

3 可靠性驱动工作流调度问题

基于以上的信誉度计算模型,笔者定义了云计算环境中可靠性驱动的工作流信誉度调度问题。在工作流应用调度中,只有其父任务完成后,任务才可以开始执行,因此,对于任务 v_i,其可用开始时间 t_i^{avail} 可表示为

$$t_i^{\text{avail}} = \max_{e(i,j) \in E} t_j^e \tag{3}$$

其中:t_j^e 表示任务 v_j 的结束时间。如果任务 v_i 不存在父任务,则其可用开始时间为 0。令 idle(i)表示资源 r_j 的空闲时间,那么,任务 v_i 的开始时间 t_i^s 和结束时间 t_i^e 可分别定义为

$$t_i^s = \max\{t_i^{\text{avail}}, \text{idle}(M(i))\} \tag{4}$$

$$t_i^e = t_i^s + |v_i| \gamma_j \tag{5}$$

其中:$M(i) = r_j$,$M(i)$ 表示任务 v_i 调度的目标资源为 v_j;γ_j 表示资源 r_j 的指令计算速度。因此,在调度 S 中,资源 r_j 完成其所有调度任务的时间 t_S^j 可定义为

$$t_S^j = \max_{i | M(i) = r_j} \{t_i^e\} \tag{6}$$

工作流应用调度的可靠性即为所有任务成功完成的概率。由于 rdr_i 表示资源 r_i 的失效率,所以,在调度 S 中,资源 r_i 能够完成其所有任务的概率可表示为

$$R_S^i = e^{-t_S^i \times rdr_i} \tag{7}$$

因此,调度 S 中工作流应用的成功率 R_S 可表示为

$$R_S = \prod_{i=1}^{m} R_S^i = e^{-\sum_{i=1}^{m} t_S^i \times rdr_i} \tag{8}$$

工作流应用可靠性驱动调度问题的目标即为最大化任务调度可靠性并最小化工作流调度执行跨度 makespan。为了最大化调度 S 的可靠性,需要最小化失效因子,失效因子为

$$\text{fail}(S) = \sum_{i=1}^{m} t_S^i \times rdr_i \tag{9}$$

因此,本文的调度问题目标可形式化为以下优化模型:

$$\min \text{fail}(S) = \sum_{i=1}^{m} t_S^i \times rdr_i \tag{10}$$

$$\min \text{makespan}(S) = \max_{r_i \in R}(t_S^i) \tag{11}$$

4 可靠性驱动调度遗传算法 RDR-GA

对于工作流应用调度问题,遗传算法可以通过在多代数调度方案中进行迭代进化选择满意解。典型的遗传算法操作步骤包括:a)创建由随机产生的解组成的初始种群,即染色体;b)评估每个解的适应度并选择下一种群的解;c)通过利用遗传算子(交叉与变异)产生新一代解;d)重复步骤 b)c)直至种群达到收敛。通常,遗传算子中使用随机化方式生成调度方案,这会导致很多无效解或算法收敛速度太慢。为了解决这一问题,笔者设计了基于评估的任务执行序列替代进化步骤,并利用基于任务优先级的 MAX-MIN 算法寻找最优映射解。算法具体设计思想如下。

4.1 问题编码

对于工作流调度问题,染色体即为调度解的编码数据结构表示。如图 1(a)给出的工作流示例,对于图 1(b)中的一种调度方案可由图 1(c)给出的二维字符串表示,该字符串的一维表示资源索引,描述任务—资源映射关系,另一维表示任务间的排序关系。然后,二维字符串被转换为任务—资源映射串 M(图 1(d)),表示长度为 $|V|$ 的向量。此时,由于其含义是相同的,任务—资源映射串拥有与映射函数相同的标志 M,即 $M(i) = r_j$ 表明任务 v_i 被调度至资源 r_j。

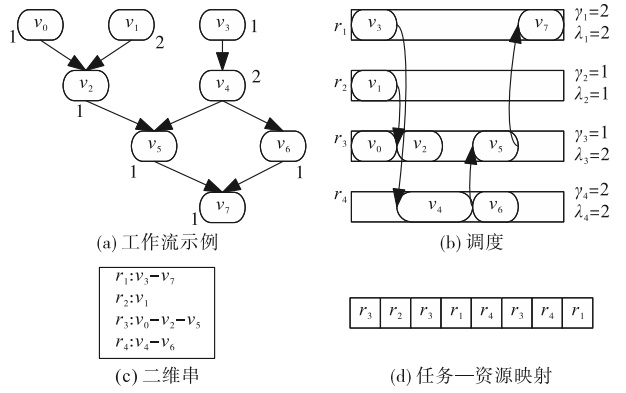


图1 调度编码

4.2 遗传交叉

遗传交叉操作通过交换两个适宜染色体可以得到更好的染色体。为了保持任务间的依赖关系,且由于对于任务—资源映射中一个好的任务执行序列在另一个任务—资源映射中并不一定是好的,所以,笔者设计的遗传交叉中仅交换两个染色体间的任务—资源映射解,两个新的子代中的任务序列通过设计的任务优先级在随后的评估步骤中得到,这样可以确保对于一个特定的任务—资源映射解中可以得到一个可行的任务执行序列。交叉操作具体为:从当前种群中以概率 p_c 随机选择染色体对,对于如图 2 中的一对染色体,在任务—资源映射串 M 中随机产生一个中断点,将串划分为上下两部分,下部分被相互交换以形成两个新的任务—资源映射子代。

4.3 遗传变异

遗传变异操作可以将搜索过程从局部最优中脱离出来,以随机改变染色体的基因结构。笔者设计了一种基于资源优先级的变异操作使得解得到突变。文献[11]的研究表明,为了最优化工作流应用执行的可靠性,拥有最小指令速度(单位指令执行时间)与失效率之积的资源在调度过程中应该具有更高的优先级,基于此结论,定义:

引理 1　资源优先级。令 $1/(\gamma_i rdr_i)$ 为资源 r_i 的优先级,S 表示所有任务被调度至具有最高优先级资源时的一种调度方案,那么,任意拥有可靠性 $R_{S'}$ 的调度 $S' \neq S$ 时,能使得 $R_{S'} < R_S$。

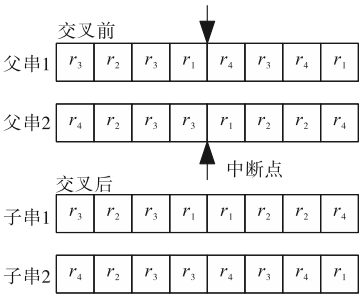


图2　遗传交叉

类似于遗传交叉操作,变异操作仅交换一个解中的任务—资源映射解,即以概率 p_m 在解中进行变异操作。变异操作随机选择解中的一个任务,然后,重新将其调度至拥有更低 $\gamma_i rdr_i$ 的任意资源上。如图 3(a)所示,任务 v_4 初始时被调度至资源 r_4 上,其 $\gamma_i rdr_i = 4$,因此,变异操作重新将其调度至 $\gamma_i rdr_i = 1$ 的资源 r_2 上。图 3(b)显示了变异后的新调度方案,该方案中工作流应用的执行跨度 makespan 和可靠性均可以得到改进。

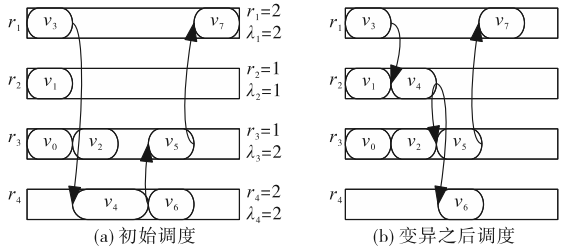


图3　遗传变异

4.4　遗传进化

遗传进化操作中,RDR-GA 算法首先调度新生成解的任务执行序列,然后,算法计算每个任务 v_i 的估计结束时间 t_i^e,从而利用式(5)评估新调度 S 的执行跨度 makespan 和失效因子。为了给出特定任务—资源映射串中一种最优任务执行序列,首先需要定义两种任务优先级,并在这两种任务优先级的基础上设计新的 MAX-MIN 策略进行优化,而为了最优化工作流的执行跨度,则需要为开始更早或对执行跨度有更大影响的任务赋予更高优先级。以下对该过程作详细说明:

定义 2　任务优先级 1(TP1)。令任务 v_i 的重要因子 $\mathrm{imprt}(i)$ 表示工作流 DAG 中以任务 v_i 为起点的最长路径长度,定义为

$$\mathrm{imprt}(i) = \begin{cases} |v_i| & v_i \text{ 为出口任务} \\ |v_i| + \max\limits_{e(i,j)\in E} \mathrm{imprt}(j) & \text{否则} \end{cases} \tag{12}$$

因此,任务 v_i 的优先级 $p(i)$ 可表示为

$$p(i) = E(\gamma) \times \mathrm{imprt}(i) - \max(t_i^{\mathrm{avail}}, \mathrm{idle}(M(i))) \tag{13}$$

其中:$E(\gamma)$ 表示所有资源的平均指令速度。

如果两个任务被调度至相同资源上,则拥有更高优先级的的任务优先被执行。任务优先级 TP1 利用所有资源的平均指令速度估计以该任务为起点的最长路径的执行时间。而在笔者设计的遗传算法中,由于之前的进化步骤中的任务—资源映射是固定的,所以对于一条路径的执行时间估计将更加准确,基于此,进一步作以下定义:

定义 3　任务优先级 2(TP2)。令以任务 v_i 为起点的最长路径的估计执行时间 $\mathrm{comp}(i)$ 为

$$\mathrm{comp}(i) = \begin{cases} |v_i| \times \gamma_j & v_i \text{ 为出口任务} \\ |v_i| \times \gamma_j + \max\limits_{e(i,k)\in E} \mathrm{comp}(k) & \text{否则} \end{cases} \tag{14}$$

其中,$M(i) = r_j$。因此,任务 v_i 的优先级 $p(i)$ 可表示为

$$p(i) = \mathrm{comp}(i) - \max(t_i^{\mathrm{avail}}, \mathrm{idle}(M(i))) \tag{15}$$

结合定义 2 和 3,笔者设计了一种基于两阶段 MAX-MIN 策略[12]的遗传进化算法,如算法 2 所示。对于每个资源,算法首先选择该资源上需要调度的下一个任务,该任务在 TP1 和 TP2 上拥有最大的优先级。然后,从资源上所有下一调度任务中,选择拥有最小结束时间的任务进行调度。对于给定的任务—资源映射串 M(映射函数),被分配至资源 r_i 的所有任务放入其调度序列 que_i 中,算法输出新的调度 S 中每个资源 r_i 的估计完成时间 t_i^s。$\mathrm{que\_ready}_i$ 为包含在资源 r_i 上等待执行而未被调度任务的队列。

算法 2　遗传进化

```
1   input: task-resource mapping string M
2   output: {t_S^i, que_i} for each resource r_i
3   foreach entry task v_j do
4     add v_j to task ready queue que_ready_M(j)
5     t_j^avail ←0
6   repeat
7     //minimum end time
8     min_end←∞
9     //task selected
10    task_sel←null
11    foreach resource r_i do
12      //MAX-MIN phase 1
13        find task v_j with the maximum priority value from que_ready_i
14      //MAX-MIN phase 2
15      compute t_j^e using equation (5)
16      if t_j^e < min_end then
17        //update minimum end time
18        min_end←t_j^e
19        //update task selected
20        task_sel←j
21    res_sel←M(task_sel)
22    remove v_task_sel from que_ready_res_sel
23    add v_task_sel to que_res_sel
24    tres_sel S = idle(res_sel) = t^e task_sel
25    foreach child task v_i of task v_task_sel do
26      compute t_j^avail using equation (3)
27      if v_i is ready to run then
28        add v_i to que_ready_M(i)
29    until every que_ready_i is empty
```

算法说明:a)行 3 ~ 5 将每个入口任务 v_j 添加至为其分配资源 $M(j)$ 的任务就绪队列中,并设置其可用开始时间为 0;b)行 13 为每个资源选择最大优先级任务;c)行 14 ~ 20 中,在所有选择的任务中,拥有最小结束时间的任务 $v_{\mathrm{task\_sel}}$ 被选择为调度任务;d)行 21 ~ 23 调度被选择任务 $v_{\mathrm{task\_sel}}$,然后行 24 更新任务完成时间和资源 $M(\mathrm{task\_sel})$ 的空闲时间;e)行 25 ~ 28 更新被调度任务的所有子任务的状态;f)行 29 重复以上步骤 b) ~ e)直到所有任务被调度完成为止。

定理 1　遗传进化算法的时间复杂度。遗传进化算法的时间复杂度为 $O(n \log n + nm + d)$。其中:m 表示资源数量;n 表示 DAG 中节点(任务)数量;d 表示有向边(依赖关系)数量。

证明　行 3 ~ 5 中初始化任务就绪队列的时间复杂度为 $O(n)$;行 6 ~ 29 为一次调度一个任务的完整迭代过程,因此,该过程将执行 n 次;行 13 为了有效地对每个资源进行排序并为其选择任务,其时间复杂度为 $O(\log n)$;行 15 ~ 20 计算任务结束时间并选择最小结束时间任务的时间复杂度为 $O(m)$;行 21 ~ 24 的时间复杂度为 $O(1)$,因此,行 6 ~ 24 的重复过程的

时间复杂度为 $O(n(\log n+m+1))$;行 25 ~ 28 更新子任务的可用任务,其时间复杂度为 $O(d)$。因此,进化算法的总时间复杂度为 $O(n+n(\log n+m+1)+d)=O(n\log n+nm+d)$。

4.5 遗传选择

遗传算法中,适应度函数的目标是对解进行度量与选择。本文的目标是在时间约束下最优化工作流应用的可靠性和执行跨度,笔者设计了以下对于调度 S 的适应度函数 $f(S)$:

$$f(S)=w_1\times\frac{\text{fail}(S)-\text{minFail}}{\text{maxFail}-\text{minFail}}+$$

$$w_2\times\frac{\text{time}(S)-\text{minTime}}{\text{maxTime}-\text{minTime}},r(w_1+w_2=1) \tag{16}$$

其中:maxFail 和 minFail 分别表示在当前代中解的最大和最小失效因子;maxTime 和 minTime 分别表示最大和最小执行跨度 makespan。可以看出,适应度函数 $f(S)$ 可以使进化算法优先选择拥有最小失效因子和最小执行跨度的解,同时,为了均衡用户需求,为两个目标分配了权重 w_1 和 w_2,因此,为下一代选择更好的解,可以在种群进化过程中对染色体以其适应度 $f(S)$ 作升序排列,然后利用常规遗传算法中的轮盘赌策略为下一代选择最优解。

5 仿真实验

5.1 实验参数

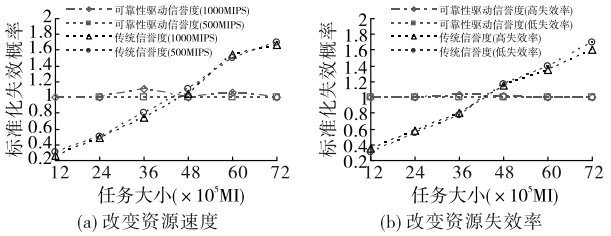
利用 WorkflowSim[13] 模拟云计算中工作流的调度环境,以评估工作流调度算法的性能。资源数量 m、平均资源速度 γ 和平均资源失效率 λ 分别设置为 $m=200$,可提供不同种类和数量的 CPU 计算周期,资源速度服从单位指令下区间 $[5\times10^{-4},10^{-3}]$/ms 的均匀分布,资源失效率服从区间 $[10^{-3},10^{-4}]$/h 的均匀分布。

利用随机 DAG 生成器模拟产生工作流应用模型,其中包括任务数量、任务节点的平均出度及平均任务大小三个参数,任务数量选取区间为[40,200],任务节点的平均出度为 2,任务大小服从区间 $[1\times10^4,15\times10^6]$/MI 的均匀分布。所有资源的初始信誉度 rdr^{initial} 设置为 10^{-3}/h,即每小时产生 10^{-3} 次失效,信誉度衰减因子 $\alpha=0.2$,权值 $w_1=w_2=0.5$,即算法给予可靠性和执行跨度同等的优先级,遗传操作中交叉概率 $p_c=0.5$,变异概率 $p_m=0.25$,遗传算法的种群规模设置为 20。

5.2 实验结果

实验 1 观察可靠性驱动的信誉度度量模型的性能。选取传统信誉度作为度量基准,传统信誉度即为成功完成任务的比例。任务大小以递增序列进行观察,设置为 $\{12,24,36,48,60,72\}\times10^5$ MI,并讨论两种场景,即改变资源速度场景和改变资源失效率场景。资源速度设置为 1 000 MIPS(快)和 500 MIPS(慢),资源失效率设置为 10^{-3}/h(高)和 10^{-4}/h(低)。

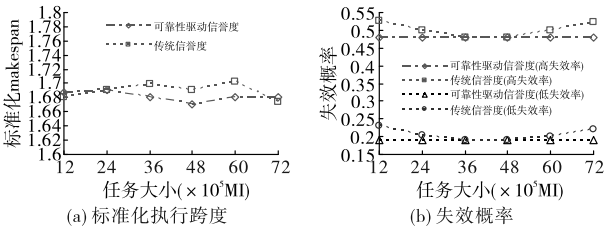
图 4 是两种信誉度度量模型的失效概率情况。可以看出,本文的可靠性驱动的信誉度的失效概率在不同的任务大小下基本维持不变,且与实际的资源失效率(标准值为 1)大致相同,而传统信誉度的失效概率仅在任务大小为 48×10^5 MI 时才接近于实际的资源失效率。另外,传统信誉度的失效概率会随着任务大小的增加而增加,同时,当资源拥有更快的速度(图 4(a))或更低的失效率(图 4(b))时,传统信誉度的失效概率将偏离实际值更多,这是由于传统信誉度的标准化失效概率服从负指数函数分布,而更快的速度或更低的失效率会导致更小的指数,从而带来更大的偏离。



(a) 改变资源速度 (b) 改变资源失效率

图4 信誉度度量模型比较

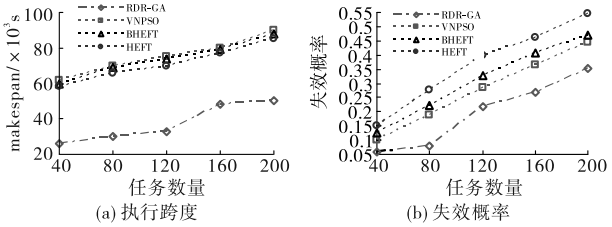
实验 2 观察不同信誉度模型对工作流调度结果的影响。实验中设置一半资源拥有实际失效率,另一半资源拥有可靠性驱动信誉度下的失效率或传统信誉度任务失效率。如图 5 所示,可以看出,两种信誉度在任务大小增加的情况下基本获得了大致相同的执行跨度 makespan(图 5(a)),然而,可靠性驱动的信誉度拥有更低的失效概率(图 5(b))。同时,传统信誉度的失效概率在任务数量减小或增大时表现更差,这是由于传统信誉度给予不同的资源失效率,这会导致任务将被调度至不可靠资源上。



(a) 标准化执行跨度 (b) 失效概率

图5 工作流执行性能比较

实验 3 观察算法与同类型启发式算法比较时工作流执行 makespan 与可靠性情况。选择 HEFT[5]、BHEFT[6] 和 VNPSO[9] 三种启发式算法作为比较基准。任务数量的变化设置为[40,200],如图 6 所示。可以看出,RDR-GA 在 makespan 和可靠性方面均优于其他算法,且比较在任务数量较小(40 个任务)和任务数量较大(200 个任务)时,RDR-GA 在 makespan 和可靠性方面可以提升约 15% 的性能,这主要是由于当任务数量越少时,对于 RDR-GA 而言,将有更多的空闲资源以供选择,这样算法可以检验每个资源以寻找最优目标资源;另一方面,HEFT、BHEFT 和 VNPSO 均为线性启发式算法,仅根据启发值检测单个资源,无法得到最优解。



(a) 执行跨度 (b) 失效概率

图6 算法性能

实验 4 观察本文算法与另一种双目标遗传算法 BGA[10] 的性能比较情况。图 7 统计了随着迭代次数的递增两种算法的平均标准化 makespan 和可靠性情况。配置工作流任务数量为 200,两种遗传算法的迭代次数设置为 1 000,从结果可以看出,RDR-GA 比较 BGA 能够以更快的速度改善工作流应用的执行跨度和可靠性,同时,在相同迭代次数时,RDR-GA 也拥有比 BGA 更高的调度解质量。

6 结束语

本文研究了云计算环境中可靠性驱动的 (下转第 1411 页)

4　结束语

在本文中,为了应对异构环境下由于木桶效应导致的任务执行时间过长,提出了一种将执行节点排序与任务划分相结合的动态算法,使得最高性能节点执行最复杂任务,次高性能节点执行次复杂任务并依此类推。在评价节点性能高低与任务相似程度的过程中都充分考虑到了不同条件下影响因子的权重不同所导致的影响。最后经过实验验证,发现本算法确实在减少任务执行时间、合理化资源利用率方面有着明显的提升效果。下一步的工作可以将本算法在处理大量数据方面的优势更加深化,并针对云平台的特点进行优化与改进,将其部署到云平台上,从而最大化地发挥本算法的特点与优势。

参考文献:

[1] 程学旗,靳小龙,杨婧,等. 大数据技术进展与发展趋势[J]. 科技导报,2016,34(14):49-59.

[2] 李国杰. 大数据研究的科学价值[J]. 中国计算机学会通信,2012,8(9):8-15.

[3] Manyika J,Chui M,Brown B,*et al*. Big data:the next frontier for innovation,competition,and productivity[EB/OL]. (2012-10-02)[2016-12-25]. http://www.mckinsey.com/Insights/MGI/Research/Techonlogy_and_Innovation/Big_data_The_next_frontier_for_innovation.

[4] 刘朵,曾锋,陈志刚,等. Hadoop 平台中一种 Reduce 负载均衡贪心算法[J]. 计算机应用研究,2016,33(9):2656-2659.

[5] Zaharia M,Konwinski A,Joseph A,*et al*. Improving MapReduce performance in heterogeneous environments[C]//Proc of USENIX Symposium on Operating Systems Design & Implementation. 2008:29-42.

[6] Snavely A,Wolter N,Carrington L. Modeling application performance by convolving machine signatures with application profiles[C]//Proc of IEEE International Workshop on Workload Characterization. 2001:149-156.

[7] Yong M,Garegrat N,Mohan S. Towards a resource aware scheduler in Hadoop[C]//Proc of the 7th IEEE International Conference on Web Services. Washington DC:IEEE Computer Society,2009:102-109.

[8] Xie Jiong,Yin Shu,Ruan Xiaojun,*et al*. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters[C]//Proc of IEEE International Symposium on Parallel & Distributed Processing Workshops and PHD Forum. 2010:1-9.

[9] Sun Xiaoyu,He Chen,Lu Ying. ESAMR:an enhanced self-adaptive MapReduce scheduling algorithm[C]//Proc of the 18th International Conference on Parallel and Distributed Systems. Washington DC:IEEE Computer Society,2012:148-155.

[10] Rasooli A,Down D. A hybrid scheduling approach for scalable heterogeneous Hadoop systems[C]//High Performance Computing,Networking Storage and Analysis. Washington DC:IEEE Computer Society,2012:1284-1291.

[11] 侯佳林,王佳君,聂洪玉. 基于异常检测模型的异构环境下 MapReduce性能优化[J]. 计算机应用,2015,35(9):2476-2481.

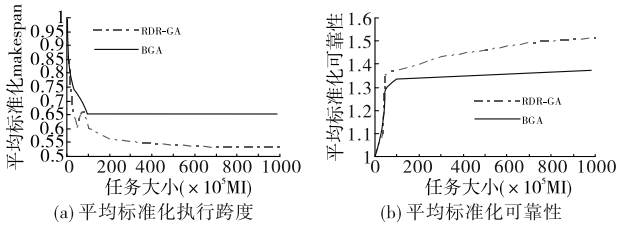
[12] Frank M,Wolfe P. An algorithm for quadratic programming[J]. Naval Research Logistics Quarterly,2006,3(1-2):95-110.

[13] 唐冲. 基于 MATLAB 的非线性规划问题的求解[J]. 计算机与数字工程,2013,41(7):1100-1102.

[14] 郑思. 大规模数据处理系统中 MapReduce 任务划分与调度关键技术研究[D]. 长沙:国防科学技术大学,2014.

[15] Massie M,Li B,Nickoles B,*et al*. Monitoring with Ganglia[M]. Sebastopol:O'Reilly Media,2012:20-63.

(上接第 1394 页)工作流调度问题,提出一种基于时间依赖的资源信誉度评估模型及算法。可靠性驱动信誉度利用任务失效率对资源信誉度进行定义,并应用指数失效模型评估任务可靠性。算法以遗传模型作为基础,以工作流执行跨度和可靠性的同步最优化作为进化目标,同时设计新的进化和评估机制,实现了较好的寻优能力。实验结果表明,算法不仅能以更精确的信誉度改进工作流应用执行可靠性,而且以比传统遗传算法更快的收敛速度得到进化更优解。



(a) 平均标准化执行跨度　　(b) 平均标准化可靠性

图7　算法性能

参考文献:

[1] Buyya R,Yeo S,Venugopal S,*et al*. Cloud computing and emerging IT platforms:vision,hype,and reality for delivering computing as the 5th utility[J]. Future Generation Computer Systems,2011,25(6):599-616.

[2] Wu Fuhui,Wu Qingbo,Tan Yusong. Workflow scheduling in cloud:a survey[J]. Journal of Supercomputing,2015,71(9):3373-3418.

[3] Juve G,Chervenak A,Deelman E,*et al*. Characterizing and profiling scientific workflows[J]. Future Generation Computer Systems,2013,29(3):682-692.

[4] 景维鹏,吴智博,刘宏伟,等. 多 DAG 工作流在云计算环境下的可靠性调度方法[J]. 西安电子科技大学学报:自然科学版,2016,43(2):83-88.

[5] Topcuoglu H,Hariri S,Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Trans on Parallel & Distributed Systems,2012,13(3):260-274.

[6] Zheng Wei,Sakellariou R. Budget-deadline constrained workflow planning for admission control[J]. Journal of Grid Computing,2013,11(4):633-651.

[7] Arabnejad H,Barbosa J G. A budget constrained scheduling algorithm for workflow applications[J]. Journal of Grid Computing,2014,12(4):665-679.

[8] 闫歌,于炯,杨兴耀. 基于可靠性的云工作流调度策略[J]. 计算机应用,2014,34(3):673-677.

[9] 马俊波,殷建平. 云计算环境下带安全约束的工作流调度问题的研究[J]. 计算机工程与科学,2014,36(4):607-614.

[10] Doğan A,Özgüner F. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems[J]. Journal of Computer,2013,48(3):300-314.

[11] Dongarra J J,Jeannot E,Saule E,*et al*. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems[C]//Proc of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures. New York:ACM Press,2007:280-288.

[12] 王岩,汪晋宽,王翠荣,等. QoS 约束的云工作流调度算法[J]. 东北大学学报:自然科学版,2014,35(7):939-943.

[13] Chen Weiwei,Deelman E. WorkflowSim:a toolkit for simulating scientific workflows in distributed environments[C]//Proc of the 8th International Conference on E-Science. New York:IEEE Press,2012:1-8.