DOI:10.7544/issn1000-1239.2016.20151175
53(9):1990-1999,2016

任务调度算法中新的自适应惯性权重计算方法

李学俊　徐　佳　朱二周　张以文
(安徽大学计算机科学与技术学院　合肥　230601)
(xjli@ahu.edu.cn)

A Novel Computation Method for Adaptive Inertia Weight of Task Scheduling Algorithm

Li Xuejun, Xu Jia, Zhu Erzhou, and Zhang Yiwen
(*School of Computer Science and Technology, Anhui University, Hefei* 230601)

Abstract　Particle swarm optimization (PSO) is a primary intelligence algorithm to solve task scheduling problem for workflow systems in cloud computing. The inertia weight is one of the most important parameters to achieve a balance between the global and local search in PSO algorithm. However, traditional adaptive inertia weight-based PSO task scheduling algorithms usually get local optimal and cause longer execution time and higher cost for scheduling plan. The traditional adaptive inertia weight does not comprehensively represent the information of particle position, and then cannot make a suitable balance between global and local search. Hence, a novel computation method for adaptive inertia weight is proposed to improve the computation method of success value of each particle. This method shows the position state of each particle more accurately and then improves the adaptability of inertia weight by comparing the fitness of each particle with the global best particle. Then a new inertia weight-based PSO algorithm is presented to solve task scheduling problem for cloud workflow systems. The novel weight can adjust the particle velocity more correctly so that the algorithm avoids premature convergence by a proper balance between local and global search. Comparing our new adaptive inertia weight with other five traditional inertia weights (viz. constant, index decreasing, linear decreasing, random, and adaptive inertia weight), the results show that our new adaptive inertia weight-based scheduling algorithm can always achieve stable convergence speed, the optimal fitness and execution cost of the scheduling plan (i.e. roughly 18% reduction of execution cost).

Key words　cloud computing; workflow; task scheduling; particle swarm optimization (PSO); inertia weight

摘　要　粒子群算法(particle swarm optimization, PSO)是解决云计算环境中工作流系统的任务调度优化问题的主流智能算法.然而基于传统自适应惯性权重的粒子群任务调度算法易陷入局部最优,导致调度方案的执行时间与费用较高.因此,通过改进单个粒子的成功值计算方法,提出了一种新的自适应惯性权重计算方法NAIWPSO(new adaptive inertia weight based particle swarm optimization).该方法通过比较每个粒子的适应度与全局最优值,可以更加精确描述粒子状态,进而提高了权重的自适应性.在新惯性权重基础上,提出了一种解决云工作流系统中任务调度优化问题的改进粒子群算法.新权重可以更准确的调整粒子速度,使算法更好地平衡粒子全局与局部搜索,避免陷入局部最优,获得执行费用更

收稿日期:2015-12-22;**修回日期**:2016-04-19
基金项目:国家自然科学基金项目(61300169);安徽省教育厅自然科学研究重点项目(KJ2016A024)
This work was supported by the National Natural Science Foundation of China (61300169) and the Natural Science Foundation of Education Commission of Anhui Province (KJ2016A024).

优的调度方案.实验表明,与5种已有惯性权重算法比较,新算法收敛稳定、适应度最低、执行费用平均减少18%.

关键词 云计算;工作流;任务调度;粒子群算法;惯性权重

中图法分类号 TP391

工作流系统能够根据用户的需求管理与优化工作流的任务调度[1].云计算是一种高效、可伸缩、高可靠性的网络资源模型[2],为用户快速分配与回收各种资源(包括网络、服务、存储、应用资源等)[3-4].随着云环境中科学计算、大规模电子商务等应用的发展,用户的服务质量(quality of service, QoS)要求不断提高,系统应用的运行费用急速上涨[5].因此,如何在满足QoS约束的前提下降低云环境中工作流运行费用是一个值得研究的问题[6].在文献[7]中,Yu等人提出了一种在网格计算环境中最小化任务执行费用的工作流调度算法.Wieczorek等人[8]使用HEFT任务调度算法较好地解决了网格计算环境中科学工作流调度问题.Xu等人[9]提出了一种在大量任务情况下的资源分配模型,有效降低了资源使用费用.云工作流管理系统可以根据用户的需求为每个任务分配合适的资源,有效降低云计算环境中工作流的运行费用[10-15].

目前云工作流管理系统中最常使用的任务调度算法为Eberhart和Shi[16]于1998年提出的粒子群算法(particle swarm optimization, PSO).该算法最初是受到鸟群、鱼群等群体动物的合作性行为启发,进而利用群体智能建立的一个随机优化启发式算法[17].PSO算法具有参数少、易实现、收敛速度快等特性,已被广泛地应用于各种优化领域.然而PSO算法没有有效地控制粒子的速度,因此无法很好地平衡粒子全局与局部搜索能力,具有局部收敛的缺陷,导致无法在有限的迭代次数中找到最优解.

为了解决局部收敛的缺陷,粒子群算法的速度更新公式增加了惯性权重这一参数,以增强平衡粒子全局与局部搜索的能力[16].目前,惯性权重大体上分为3类:常数或随机型惯性权重[16-18]、线性或指数变化惯性权重[19-21]、自适应性惯性权重[22].固定常数值惯性权重[16]实现简单,在算法执行过程中,惯性权重固定为一个常数,导致算法容易陷入局部最优.因此Eberhart和Shi[18]提出了随机惯性权重,以优化目标动态变化的问题.Shi和Eberhart[19-20]又提出线性减少惯性权重,以提高粒子群算法的求精能力,该算法如果在初期粒子搜索不到较好的位置点,易陷入局部最优.在线性减少惯性权重的基础上,Chatterjee和Siarry[21]提出了指数减少惯性权重,在一定程度上避免了算法陷入局部最优.然而,指数值的确定是一个难点,而且算法的计算量较大,不适合复杂优化问题求解.

综合上述非自适应惯性权重,Nickabadi等人[22]提出了基于自适应惯性权重(adaptive inertia weight, AIW)的粒子群算法(adaptive inertia weight based particle swarm optimization, AIWPSO).该算法可以在算法迭代过程中根据整个粒子群中粒子的位置状态信息对惯性权重进行动态地调整,提高PSO算法的寻优能力.然而,该参数中的成功值计算方法只考虑了单个粒子在不同迭代次数中位置的优劣,造成了自适应机制无法精确地调整惯性权重,导致PSO算法速度调节不够精确,易陷入局部最优.因此,本文通过改进传统权重中的成功值计算方法,提出了一种新的自适应惯性权重(new adaptive inertia weight, NAIW);并将基于新自适应惯性权重的粒子群算法(new adaptive inertia weight based particle swarm optimization, NAIWPSO)运用到解决云工作流任务调度优化问题.实验结果表明,NAIWPSO任务调度算法在算法收敛速度、适应度和执行费用3方面均优于其余5种(固定、线性、指数、随机、传统自适应)惯性权重的粒子群算法.

1 自适应惯性权重

本节首先介绍了表示粒子位置状态优劣的适应度计算方法;然后分析了传统自适应惯性权重在成功值计算方面存在粒子位置状态表示不全面的不足;最后,提出了一种新的自适应惯性权重,改进了成功值计算公式.该权重可以更加精确地调整粒子速度,使算法更好地平衡粒子全局与局部搜索,达到避免算法局部收敛、搜索到最优值的目的.

1.1 适应度

适应度是评价任务调度方案执行时间和费用的一项综合指标[2],表示粒子群算法中粒子位置状态的优劣,如式(1)所示.适应度可以全面衡量该调度

方案所需的时间与费用.适应度越高的调度方案,所需时间与费用的开销也就越大,粒子位置状态就越不好;相反,开销越小,粒子位置状态就越好.

$$fitness = (f_1 \times cost_{S_k}) + \left(f_2 \times 10 \times cost_{S_k} \times \frac{makespan_{S_k}}{deadline}\right) + \sum_{m=1}^{M} wastetime. \quad (1)$$

适应度的计算分为3个部分:1)完工时间未超出截止期限时($f_1=1, f_2=0$)的虚拟机运行费用;2)完工时间超出截止期限时($f_1=0, f_2=1$)的虚拟机运行费用;3)各虚拟机的空闲时间 $wastetime$ 之和.其中,$makespan_{S_k}$ 表示调度方案 S_k 中所有任务的最大完工时间;$deadline$ 表示整个工作流的截止期限,可以在 $deadline_{\min} \sim deadline_{\max}$ 之间变化.$deadline_{\min}$ 与 $deadline_{\max}$ 的计算方法如式(2)、式(3):

$$deadline_{\min} = \min\{makespan_{S_k}\}; \quad (2)$$

$$deadline_{\max} = \max\{makespan_{S_k}\}. \quad (3)$$

在适应度计算式(1)中,$cost_{S_k}$ 为某一调度方案 S_k 的费用值,其计算方法如式(4)所示:

$$cost_{S_k} = \sum_{m=1}^{M} c_{\mathrm{d}}^m t^m x_{\mathrm{d}}^m + \sum_{m=1}^{M} (p_{\mathrm{r}}^m + c_{\mathrm{r}}^m t^m) x_{\mathrm{r}}^m, \quad (4)$$

其中,等号右边的第1项是所有按需(on-demand)购买虚拟机的费用,第2项是所有按预定(reserved)购买虚拟机的费用.这种费用值计算方法允许调度方案同时考虑现实中最常见的按需购买与按预定购买2种虚拟机购买方式.其中,c_{d}^m 和 c_{r}^m 分别为虚拟机 m 按需购买和按预定购买每小时的费用,x_{d}^m 表示虚拟机 m 的购买方式是按需购买,x_{r}^m 表示虚拟机 m 的购买方式是按预定购买,t^m 是虚拟机 m 的运行时间,p_{r}^m 为虚拟机 m 按预定购买方式时客户需要提前交付的预定金.

1.2 传统的自适应惯性权重

在文献[22]中,Nickabadi等人提出了一种自适应性惯性权重AIW,该权重首先计算单个粒子的成功值(如式(5)所示),之后计算整个粒子群的成功率(如式(6)所示),最后更新惯性权重(如式(7)所示).

$S(i,t)$ 是粒子 i 在第 t 次迭代过程中的成功值.其中,$pbest_i^t$ 是粒子 i 在第 t 次迭代时的最优位置,$fitness(pbest_i^t)$ 是粒子 i 在第 t 次迭代时最优位置的适应度.在当前粒子所处位置的适应度比上次最优位置小时,该粒子在本次迭代过程中的成功值设为1,表示该粒子在本次搜索中所移动到的位置比上次好,即搜索成功;反之,当前粒子所处位置的适应度等于上次最优位置时,即在先后2次迭代过程中,该粒子的最优位置适应度没有改变,成功值设为0,表示该粒子在本次搜索中所移动到的位置比上次最优位置差,即搜索失败.

$$S(i,t) = \begin{cases} 1, & fitness(pbest_i^t) < fitness(pbest_i^{t-1}); \\ 0, & fitness(pbest_i^t) = fitness(pbest_i^{t-1}). \end{cases} \quad (5)$$

$P_{\mathrm{S}}(t)$ 是粒子群在第 t 次迭代时的成功率,表示本次迭代位置比上次好的粒子在粒子群中所占比重.其中,$\sum_{i=1}^{n} S(i,t)$ 是所有粒子的成功值之和,n 表示整个粒子群的粒子个数.

$$P_{\mathrm{S}}(t) = \sum_{i=1}^{n} S(i,t)/n. \quad (6)$$

$w(t)$ 是第 t 次迭代时的惯性权重,用于调整每次迭代时的粒子初速度;$v_i(t)$ 是第 t 次迭代时粒子 i 的速度,$v_i(t-1)$ 为上次迭代时粒子 i 的速度;r_1, r_2 为介于0,1之间的随机数;c_1, c_2 为学习因子;p_i^t 与 p_{g}^t 分别为粒子的个体极值以及种群的全局极值;x_i^t 代表粒子 i 在第 t 次迭代的位置.

$$w(t) = (w_{\max} - w_{\min}) P_{\mathrm{S}}(t) + w_{\min}, \quad 0 \leqslant w_{\min} \leqslant w_{\max} \leqslant 1, \quad (7)$$

$$v_i(t) = w(t) v_i(t-1) + r_1 c_1 (p_i^t - x_i^t) + r_2 c_2 (p_{\mathrm{g}}^t - x_i^t). \quad (8)$$

算法AIWPSO产生的调度方案执行时间和费用比普通PSO算法高.原因在于权重AIW中成功值计算公式 $S(i,t)$ 只考虑了2种情况,即当前粒子的适应度优于上次,成功值则为1,反之则为0.由于该方法没有全面分析粒子的位置状态信息,即没有考虑全局最优位置,导致了成功率(式(6))以及惯性权重值(式(7))的计算不够准确.因此,该方法无法在每次迭代时精确地调整惯性权重值,造成算法在迭代过程中不能有效平衡粒子的全局与局部搜索,使算法易陷入局部最优,无法找到最优解.

1.3 新的自适应惯性权重

针对传统惯性权重中单个粒子位置状态描述不全面的问题,本文在成功值计算方法中增加每个粒子的适应度与全局最优值比较,以更加精确地描述粒子所处的位置状态.权重NAIW中改进后的成功值如式(9)所示.在式(5)中每个粒子的当前适应度与上次迭代比较的基础上,式(9)中 $S(i,t)$ 加入了全局最优值 $globalbest^t$.新的 $S(i,t)$ 计算方法的取值扩充为3种情况:1,0.5,0.该方案基于对粒子位置状态更加细化的评价使得在粒子将要陷入局部最优解时,通过与全局最优适应度的比较来更加准

确地得出 $S(i,t)$.精确的 $S(i,t)$可以提高粒子群的成功率(式(6))的准确性和惯性权重(式(7))的自适应性.因此,算法可以更准确地调整粒子速度,以平衡粒子的全局与局部搜索能力,避免算法陷入局部最优.

$$S(i,t)=\begin{cases}1, & fitness(pbest_i^t)<fitness(pbest_i^{t-1})\\ & \text{且 } fitness(pbest_i^t)<globalbest^t;\\ 0.5, & fitness(pbest_i^t)<fitness(pbest_i^{t-1})\\ & \text{且 } fitness(pbest_i^t)\geqslant globalbest^t;\\ 0, & fitness(pbest_i^t)=fitness(pbest_i^{t-1}).\end{cases} \tag{9}$$

算法 NAIWPSO 中的权重 NAIW 会随着粒子群的不同搜索状态而改变,可以动态改变粒子的速度.当粒子群的成功率 $P_S(t)$较大时,说明整个粒子群离最优值较远,此时需要粒子以较大的初速度进行全局搜索,通过式(7)和式(9)会提高速度更新式(8)中的 $w(t)$的值,进而可以使粒子能够以较大的初速度接近最优值.当粒子群的成功率 $P_S(t)$较小时,说明整个粒子群已经接近最优值,通过式(7)和式(9)使 $w(t)$值相应地减小,这样可以使粒子搜索时的速度下降,进而可以逐步精化最优解,保证了算法的精确度.综上,通过新的成功值计算公式,结合成功率与惯性权重更新公式,可以根据当前粒子群的位置状态不断改变粒子的速度,进而动态平衡算法的全局搜索与局部搜索能力.

2 基于新自适应惯性权重的粒子群任务调度算法

基于新自适应惯性权重 NAIW 的粒子群任务调度算法 NAIWPSO 用于解决云工作流任务调度.该算法分为 2 部分:行①~⑦是对调度方案 S_k 的各项参数初始化,行⑧~㉒是调度算法对调度方案进行搜索.其中,行⑮~㉑使用权重 NAIW 更新调度方案及搜索速度,提高任务调度算法搜索最优调度方案的性能.

算法 1. 基于新自适应惯性权重的粒子群任务调度算法.

输入:算法迭代次数 *Iteration*、任务 Tasks、虚拟机 VMs、截止期限 Deadline;

输出:最优调度方案 S_{best}.

① for $i=1$ to k do
② 随机初始化任务调度方案 S_i 及其搜索速度 v_i;
③ end for
④ for $i=1$ to k do
⑤ 计算任务调度方案 S_i 的适应度;
⑥ end for
⑦ 从k 个任务调度方案中选出适应度最小的全局最优调度方案;
⑧ for $i=1$ to *Iteration* do
⑨ 根据搜索速度更新所有任务调度方案;
⑩ 为任务调度方案中的每一个任务重新分配虚拟机;
⑪ for $j=1$ to k do
⑫ 计算任务调度方案 S_i 的适应度与费用(式(1)、式(4));
⑬ end for
⑭ 从k 个任务调度方案中选出适应度最小的全局最优调度方案;
⑮ for $j=1$ to k do
⑯ 计算任务调度方案 S_j 的成功值 $Svalue_j$(式(9));
⑰ $Scount=Scount+Svalue_j$;
⑱ end for
⑲ 计算成功率(式(6));
⑳ 根据成功率更新惯性权重值(式(7));
㉑ 更新每一个调度方案的搜索速度(式(8));
㉒ end for
㉓ return S_{best}.

算法 1 首先初始化调度方案及其搜索速度以及算法所需各项参数(行①~⑦).接着根据各调度方案搜索速度更新调度方案,并将任务分配给相应虚拟机(行⑧~⑩);之后为每个调度方案计算费用与适应度,并更新全局最优调度方案(行⑪~⑭).然后根据权重 NAIW 得出最新惯性权重值 w(行⑮~⑳),再根据新的 w 更新各调度方案搜索速度(行㉑);其中 $Svalue_i$ 表示第 i 个粒子的成功值,$Scount$ 表示成功值之和.当算法执行到迭代次数后,最终得出最优任务调度方案(行㉒),否则返回行⑧继续迭代.该调度算法通过权重 NAIW,可以很好地平衡各粒子全局搜索和局部搜索,避免算法陷入局部最优,获得最优任务调度方案.假定调度方案的数量为 N,算法迭代次数为 D,任务数为 T,则传统任务调度算法 PSO 的时间复杂度为 $O(N\times D\times T)$[23-24].本文算法 NAIWPSO 改进了惯性权重的计算方法,对算法计算量没有影响,因此算法复杂度也是 $O(N\times D\times T)$.

3 算法示例分析

首先给出了一个云工作流示例,然后比较分析传统自适应惯性权重的粒子群任务调度算法 AIWPSO 以及本文所提出的任务调度算法 NAIWPSO,所生成调度方案的执行时间与费用.

工作流由若干个相互依赖的任务组成[1]. 工作流通常使用有向无环图(directed acyclic graph, DAG)来表示任务的执行先后顺序以及任务之间的相互依赖关系[2, 25]. 在 DAG 图中每一个节点都代表工作流中的 1 个任务,节点之间的连线代表任务之间的依赖关系. 图 1 反映了 1 个工作流中 5 个任务的执行先后顺序:任务 B 和 C 必须在任务 A 执行完成后才能开始执行;而任务 D 必须在任务 C 执行完成后才能开始执行;当最后 1 个任务 E 执行完成后,代表该工作流全部完成. 假定各任务在小型机上的执行时间是中型机的 1.3 倍,是大型机的 1.6 倍. 同时,小型机、中型机和大型机的每单位时间价格(USD)分别为:0.12, 0.24, 0.48. 图 1 中各任务在 3 种虚拟机上的执行时间如表 1 所示. 下面将分别使用传统自适应惯性权重的粒子群任务调度算法 AIWPSO 以及本文所提出的任务调度算法 NAIWPSO 对图 1 表示的工作流进行任务调度.

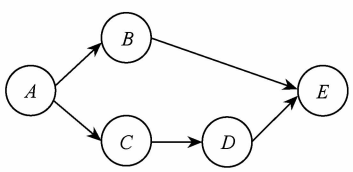

Fig. 1 Example of workflow.

图 1 工作流示例

Table 1 The Execution Unit Time of Tasks in VMs

表 1 任务在 3 种虚拟机上执行所需的单位时间

Unit Time

Virtual Machine		Task				
Type	Speed	*A*	*B*	*C*	*D*	*E*
Small	1.0	3.2	8.0	4.8	1.6	4.8
Middle	1.3	2.6	6.5	3.9	1.3	3.9
Large	1.6	2.0	5.0	3.0	1.0	3.0

假设云环境中 3 台虚拟机(小型、中型、大型各 1 台)执行图 1 中的工作流. 图 2(a)为使用任务调度算法 AIWPSO 产生的调度方案,执行时间为 16.7 单位时间,费用为 4.3USD;图 2(b)展示了本文所使用的算法 NAIWPSO 所得到的调度方案,执行时间为 14.6 单位时间,费用为 3.98USD. 从图 2 可以看出,算法 NAIWPSO 所得到的调度方案相比算法 AIWPSO,执行时间减少了 2.1 单位时间,费用减少了 0.32USD. 因此,新算法 NAIWPSO 相比 AIWPSO 能够有效降低云环境下任务调度的执行时间与费用.

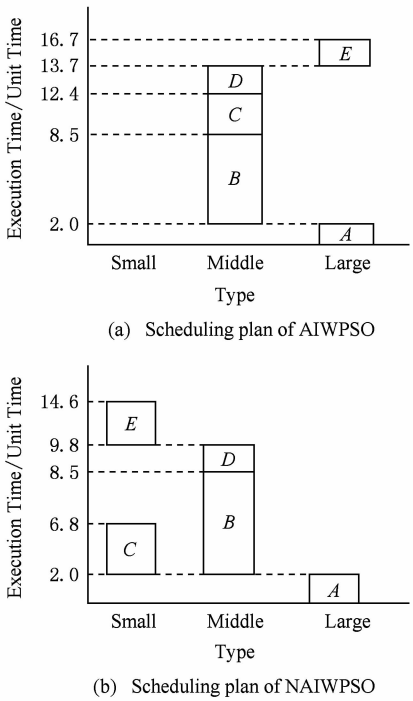

Fig. 2 Scheduling plan of two algorithms.

图 2 2 个算法的调度方案

4 实验分析

本节首先详细给出了实验环境及算法参数设置;然后从算法收敛、适应度、执行费用 3 个方面将本文算法 NAIWPSO 与其他 5 种已有算法进行了对比分析;最后,为了对不同规模工作流的截止期限约束设置提供依据,给出了算法 NAIWPSO 生成调度方案的执行费用随截止期限的变化情况.

4.1 实验环境与算法参数

本文实验采用 Matlab 2010 编写,运行在环境为 Intel core i3 3.0 GHz CPU、2 GB 内存的 PC 机上. 为了评价和分析文中算法的性能,将常数值(constant)、指数减小(index-decreasing)、线性(linear decreasing)、随机(random)、传统自适应(AIW)、新自适应(NAIW)这 6 种惯性权重应用到 PSO 任务调度算法中,对应的 PSO 算法分别表示为 CPSO, IPSO, LPSO, RPSO,

AIWPSO,NAIWPSO. 实验所使用的亚马逊 EC2(Amazon EC2①)计价模型与虚拟机运行速度[2]如表 2 所示:

Table 2 Pricing Model of Amazon

表 2 亚马逊计价模型

Virtual Machine		On-Demand		Reserved
Type	Speed	Per-Term Cost/USD	Per-Unit Time Cost/USD	Per-Unit Time Cost/USD
Small	1.0	97.5	0.07	0.12
Middle	1.3	390.0	0.28	0.48
Large	1.6	780.0	0.56	0.96

实验对象工作流采用 DAG 图方式随机生成,每个工作流总任务数为 50～300,单个任务的负载量是 30～3 000 范围内的随机值[26]. PSO 算法中粒子数量为 30,虚拟机数量为 4,学习因子 c_1 与 c_2 均为 2[2],惯性权重的最小值与最大值分别为 0,1[22]. 各算法均取不同任务值情况下运行 100 次的平均值.

4.2 实验结果与分析

本节从算法收敛、适应度、执行费用 3 个方面比较本文算法 NAIWPSO 与其他 5 种算法. 算法收敛表示算法找到最终解的速度,适应度是调度方案在时间、执行费用、虚拟机使用效率 3 个方面的综合评价标准,执行费用为调度方案执行所需费用的虚拟机计算费用.

4.2.1 算法收敛

6 种算法生成调度方案的收敛情况表 3 所示. 任务数为 50 和 300 时,生成调度方案适应度的 6 种算法收敛情况如图 3～4 所示. 从表 4 可以看出,本文算法 NAIWPSO 在不同任务数情况下得出最优调度方案的迭代次数较为稳定,整体上优于传统的自适应算法 AIWPSO. 从图 3 和图 4 可以看出,算法 NAIWPSO 生成调度方案的适应度均优于其他 5 种算法,而且任务规模越大,优化效果越显著.

Table 3 Convergence for Six Algorithms in Different Tasks

表 3 不同任务数情况下 6 种算法收敛情况

Number of Tasks	Number of Iterations					
	CPSO	IPSO	LPSO	RPSO	AIWPSO	NAIWPSO
50	80	10	40	40	85	70
100	65	8	15	15	90	65
150	25	7	95	45	70	68
200	30	7	40	25	100	55
250	25	10	100	45	80	65
300	75	15	40	40	85	51

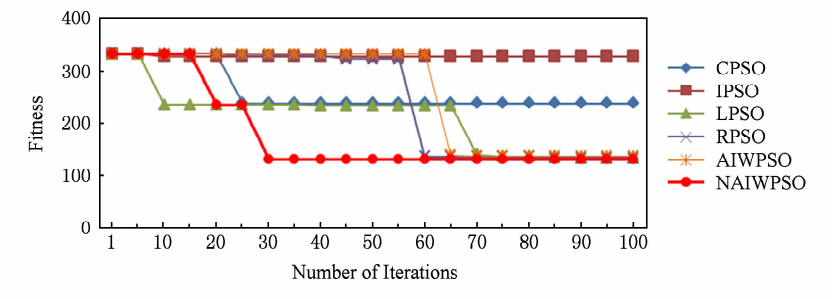

Fig. 3 Convergence of six algorithms for 50 tasks.

图 3 任务数为 50 时算法收敛情况

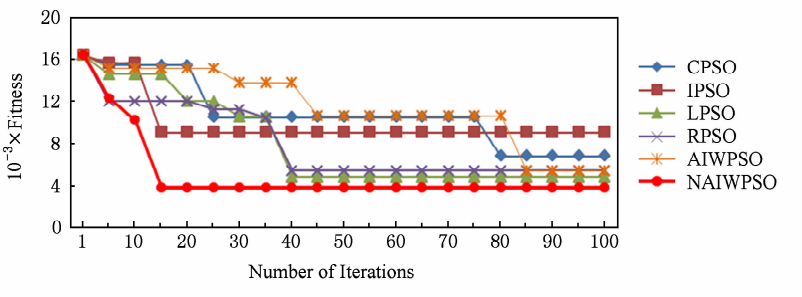

Fig. 4 Convergence of six algorithms for 300 tasks.

图 4 任务数为 300 时算法收敛情况

① http://aws.amazon.com/cn/ec2/pricing

4.2.2 适应度

在传统自适应惯性重的成功值计算方法的基础上,扩充了适应度优于上次且不优于全局最优值时取值0.5的情况,该值是根据经验设定的.为了说明取值的合理性,任务数取50～300,成功值取0.1～1.0,算法NAIWPSO的调度方案适应度变化如图5所示.本文算法NAIWPSO中成功值计算方法(式(9))第2种情况取值为0.5,与其他5种算法生成调度方案的适应度如图6所示.

从图5可以看出,对于不同任务规模的工作流,成功值取值为0.5时,算法NAIWPSO所得任务调度方案的适应度均最低.因此本文算法NAIWPSO中改进的成功值计算方法(式(9))第2种情况取值为0.5是合理的.图6中算法NAIWPSO生成调度方案的适应度均优于其他5种算法.任务数为50时,AIWPSO与NAIWPSO算法的适应度十分接近;但随着任务数量的增长,2种算法所得调度方案的适应度差距不断增大.因此,本文算法更适合于大规模任务调度优化.

4.2.3 执行费用与截止期限

6种算法产生的调度方案的执行费用如图7所示.为了给不同任务数情况下截止期限的最大值和最小值确定提供依据,任务数为50～300时,通过式(2)、式(3)计算截止期限的最小值与最大值如表4

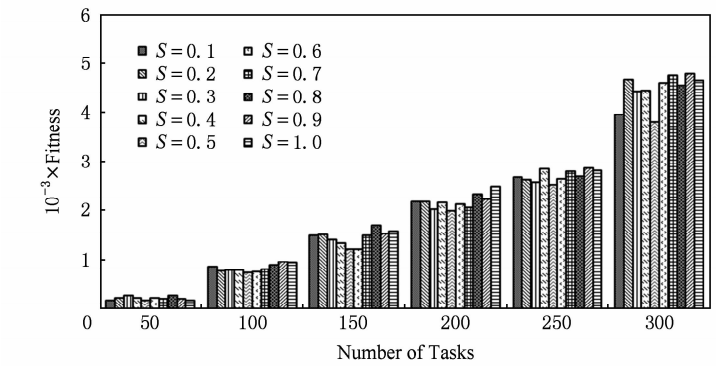

Fig. 5 Fitness of success value from 0.1 to 1.0.

图5 不同成功值的适应度

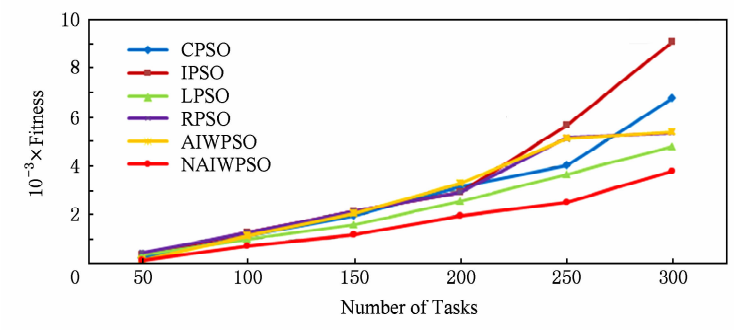

Fig. 6 Fitness of six algorithms.

图6 6种算法的适应度

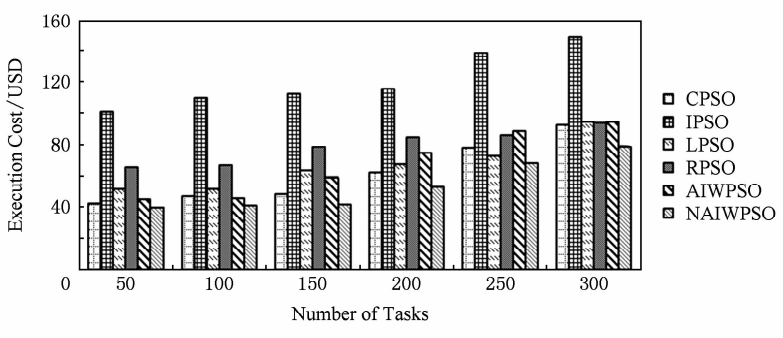

Fig. 7 Execution cost of six algorithms.

图7 6种算法的执行费用

所示,任务调度算法 NAIWPSO 的调度方案费用随截止期限的变化情况如图 8 所示. 图 7 中,随着任务量在 50～300 范围内变化,本文算法 NAIWPSO 所生成的调度方案费用值均最低,并且 NAIWPSO 算法的调度方案费用值与其他 5 种算法的差值越来越大. 这说明 NAIWPSO 算法更加适用于任务数较多的情况. 这是因为在任务数较小的情况下,各算法均进行小范围的局部搜索,使得权重 NAIW 对粒子全局与局部搜索能力的平衡优势无法体现;然而随着问题规模的增大和任务数的增加,更能体现 NAIW 权重的自适应性,更好地平衡全局与局部搜索.

Table 4 The Minimum and Maximum Value of Deadline

表 4 截止期限的最小值与最大值 Unit Time

Number of Tasks	Deadline	
	$deadline_{min}$	$deadline_{max}$
50	4.2	9.8
100	9.2	20.3
150	15.9	32.0
200	21.4	45.5
250	26.1	62.0
300	30.4	72.1

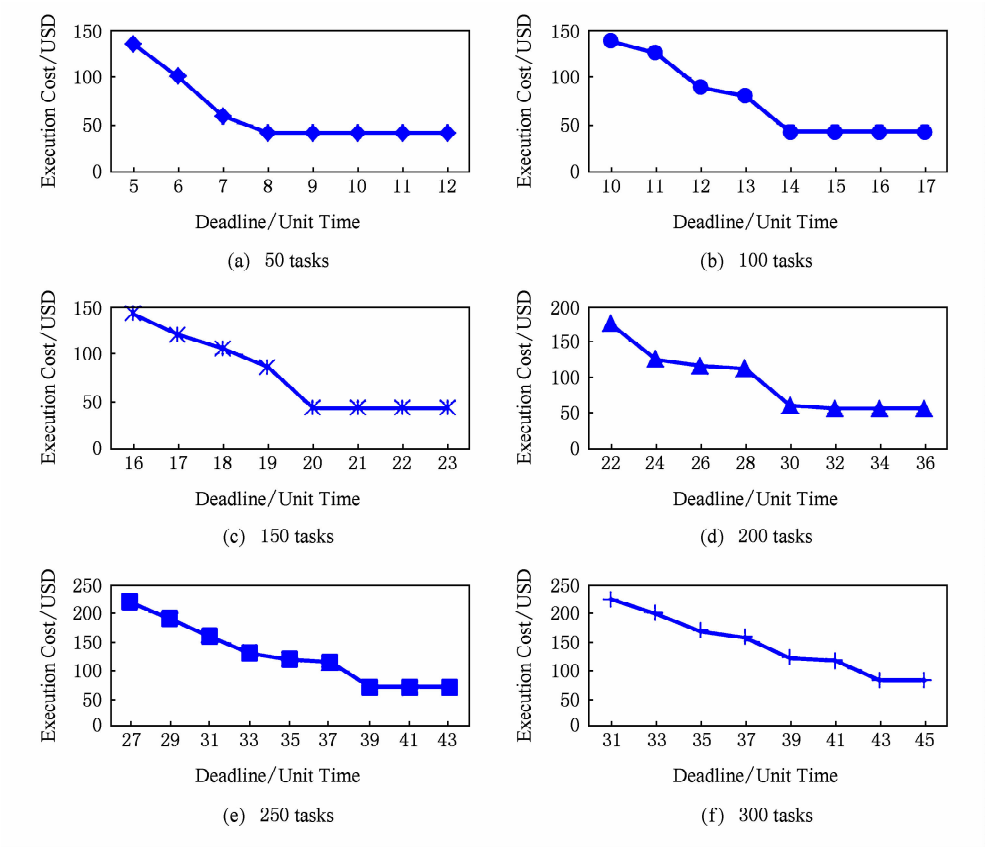

Fig. 8 Execution cost with deadline constraint for different tasks.

图 8 不同任务规模中执行费用随截止期限的变化

从图 8 可以看出,当截止期限接近最小值时,算法可能没有合适的最优调度方案,这是因为当截止期限过小时,算法无法找到最优调度方案;当截止期限在最小与最大之间时,调度方案的费用值逐渐下降;当截止期限增长到一定范围时,调度方案费用值的执行费用都趋于稳定,这是由于在截止期限的值增长到某一时刻时所有任务都已被分配到费用最优的虚拟机上. 本实验为云计算用户设置截止期限提供参考,例如任务数为 300 时,实验所得截止期限的最小值和最大值分别是 31 和 45.

5 结束语

本文针对 PSO 算法中自适应惯性权重的成功值计算方法进行了改进,提出了新的自适应惯性权重,并设计了基于新惯性权重的任务调度算法 NAIWPSO. 新自适应惯性权重中的成功值计算方法比较了单粒子与全局最优粒子,可以更加精确地描述粒子所处的位置状态,从而提高权重的自适应性. 因此,新自适应惯性权重可以更准确地调整粒子

速度以平衡粒子的全局与局部搜索,使算法避免陷入局部最优,获得时间与费用更优的调度方案.实验结果表明,新的任务调度算法 NAIWPSO 算法收敛稳定、适应度和执行费用均优于其他 5 种算法.然而,本文 DAG 图表示的工作流模型中仅考虑了顺序、选择和循环 3 种任务关系,后续工作需要研究并发、互斥等不同任务关系对调度算法的影响.

参 考 文 献

[1] Dellman E, Gannon D, Shields M, et al. Workflows and e-science: An overview of workflow system features and capabilities [J]. Future Generation Computer Systems, 2008, 25(5): 528-540

[2] Netjinda N, Sirinaovakul B, Achalakul T. Cost optimal scheduling in IaaS for dependent workload with particle swarm optimization [J]. The Journal of Supercomputing, 2014, 68(3): 1579-1603

[3] Shi Xuelin, Xu Ke. Utility maximization model of virtual machine scheduling in cloud environment [J]. Chinese Journal of Computers, 2013, 36(2): 252-262 (in Chinese)
(师雪霖, 徐恪. 云虚拟机资源分配的效用最大化模型[J]. 计算机学报, 2013, 36(2): 252-262)

[4] Wang Peng, Huang Yan, Li Kun, et al. Load balancing degree first algorithm on phase space for cloud computing cluster [J]. Journal of Computer Research and Development, 2014, 51(5): 1095-1107(in Chinese)
(王鹏, 黄焱, 李坤, 等. 云计算集群相空间负载均衡度优先调度算法研究[J]. 计算机研究与发展, 2014, 51(5): 1095-1107)

[5] Wang Qiang, Li Xiongfei, Wang Jing. A data placement and task scheduling algorithm in cloud computing [J]. Journal of Computer Research and Development, 2014, 51(11): 2416-2426(in Chinese)
(王强, 李雄飞, 王婧. 云计算中的数据放置与任务调度算法[J]. 计算机研究与发展, 2014, 51(11): 2416-2426)

[6] Lee Y, Han H, Zomaya A, et al. Resource-efficient workflow scheduling in clouds [J]. Knowledge-Based Systems, 2015, 80(5): 153-162

[7] Yu J, Buyya R, Tham C. Cost-based scheduling of scientific workflow applications on utility grids [C] //Proc of the 1st IEEE Int Conf on e-Science and Grid Computing. Piscataway, NJ: IEEE, 2005: 1-8

[8] Wieczorek M, Prodan R, Fahringer T. Scheduling of scientific workflows in the ASKALON grid environment [J]. ACM SIGMOD Record, 2005, 34(3): 56-62

[9] Xu J, Liu C, Zhao X. Resource planning for massive number of process instances [C] //Proc of the Move to Meaningful Internet Systems: OTM 2009. Berlin: Springer, 2009: 219-236

[10] Wu Z, Liu X, Ni Z, et al. A market-oriented hierarchical scheduling strategy in cloud workflow systems [J]. The Journal of Supercomputing, 2013, 63(1): 256-293

[11] Hoffa C, Mehta G, Freeman T, et al. On the use of cloud computing for scientific workflows [C] //Proc of the 4th IEEE Int Conf on eScience. Piscataway, NJ: IEEE, 2008: 640-645

[12] Sheng J, Wu W. Scheduling workflow in cloud computing based on hybrid particle swarm algorithm [J]. Telkomnika Indonesian Journal of Electrical Engineering, 2012, 10(7): 1560-1566

[13] Pandey S, Wu L, Guru S, et al. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments [C] //Proc of the 24th IEEE Int Conf on Advanced Information Networking and Applications. Piscataway, NJ: IEEE, 2010: 400-407

[14] Tao Q, Chang H, Yi Y, et al. QoS constrained grid workflow scheduling optimization based on a novel PSO algorithm [C] //Proc of the 8th Int Conf on Grid and Cooperative Computing. Piscataway, NJ: IEEE, 2009: 153-159

[15] Wu Z, Ni Z, Gu L, et al. A revised discrete particle swarm optimization for cloud workflow scheduling [C] //Proc of 2010 Int Conf on Computational Intelligence and Security. Piscataway, NJ: IEEE, 2010: 184-188

[16] Shi Y, Eberhart R. A modified particle swarm optimizer [C] //Proc of 1998 IEEE Int Conf on Evolutionary Computation. Piscataway, NJ: IEEE, 1998: 760-766

[17] Eberhart R, Shi Y. Particle swarm optimization: Developments, applications and resources [C] //Proc of 2001 IEEE Int Conf on Evolutionary Computation. Piscataway, NJ: IEEE, 2001: 81-86

[18] Eberhart R, Shi Y. Tracking and optimizing dynamic systems with particle swarms [C] //Proc of 2001 Congress on Evolutionary Computation. Piscataway, NJ: IEEE, 2001: 94-100

[19] Shi Y, Eberhart R. Empirical study of particle swarm optimization [C] //Proc of 1999 Congress on Evolutionary Computation. Piscataway, NJ: IEEE, 1999: 1945-1950

[20] Eberhart R, Shi Y. Comparing inertia weights and constriction factors in particle swarm optimization [C] //Proc of 2000 Congress on Evolutionary Computation. Piscataway, NJ: IEEE, 2000: 84-88

[21] Chatterjee A, Siarry P. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization [J]. Computers & Operations Research, 2006, 33(3): 859-871

[22] Nickabadi A, Ebadzadeh M, Safabakhsh R. A novel particle swarm optimization algorithm with adaptive inertia weight [J]. Applied Soft Computing, 2011, 11(4): 3658-3670

[23] Ho S, Lin H, Liauh W, et al. OPSO: Orthogonal particle swarm optimization and its application to task assignment problems [J]. IEEE Trans on Systems, Man and Cybernetics, Part A: Systems and Humans, 2008, 38(2): 288-298

[24] Gong M, Wu Y, Cai Q, et al. Discrete particle swarm optimization for high-order graph matching [J]. Information Sciences, 2016, 328(1): 158-171

[25] Rimal B, Jukan A, Katsaros D, et al. Architectural requirements for cloud computing systems: An enterprise cloud approach [J]. Journal of Grid Computing, 2011, 9(1): 3-26

[26] Fard H, Prodan R, Fahringer T. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments [J]. IEEE Trans on Parallel and Distributed Systems, 2013, 24(6): 1203-1212

Li Xuejun, born in 1976. PhD, associate professor. Member of China Computer Federation. His main research interests include workflow systems, cloud computing and intelligent software.

Xu Jia, born in 1992. Master candidate. His current research interests include intelligent algorithm and workflow systems (xujia_ahu@foxmail.com).

Zhu Erzhou, born in 1981. PhD, associate professor. His current research interests include program analysis, binary translation, compiling technology, virtualization and cloud computing.

Zhang Yiwen, born in 1976. PhD, associate professor. His current research interests include service computing, cloud computing and evolutionary algorithm.