论文题目

Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing

论文作者

Ahmad M. Manasrah and Hanan Ba Ali

发表期刊信息

Wireless Communications and Mobile Computing Volume 2018, Article ID 1934784, 16 pages

技术问题

本文提出了一种解决工作流调度问题的混合GA-PSO算法，通过结合两种算法的优势来解决工作流调度问题。针对其他算法评估了所提算法的效率，证明其在解决云环境中的工作流调度问题方面的有效性——所提出的算法减少了总的完工时间执行时间，并以最小的总货币成本平衡VM上的负载。

现实背景

现如今对计算和大量存储资源的需求正在快速增长。因此，云计算受到关注，因为作为软件即服务（SaaS），基础架构即服务（IaaS）和平台即服务（PaaS）提供给用户的高性能计算服务和设施。在一个任务可以执行之前，依赖任务必须首先完成其执行的意义上，可以将各种应用程序建模为具有它们之间的依赖性的一组任务的工作流应用程序。工作流程应用正在一系列领域中使用，例如天体物理学，生物信息学以及灾害建模和预测。此外，最近通过在单一解决方案中结合

各种方法和技术，出现了复杂的科学应用等复杂问题。对于这种需求，这种类型的应用程序已经在超级计算机，集群和网格上执行。幸运的是，随着云的出现，这样的工作流应用程序在云中执行。

作者思路

描述了工作流调度中的问题描述和最新技术，以及在IaaS平台上应用现有公共调度算法时的挑战。设计遵循了工作流调度算法和所提算法的定义。详细介绍了云中多目标调度问题的性能评估

解决方案

构建模型

GA-PSO算法的主要步骤

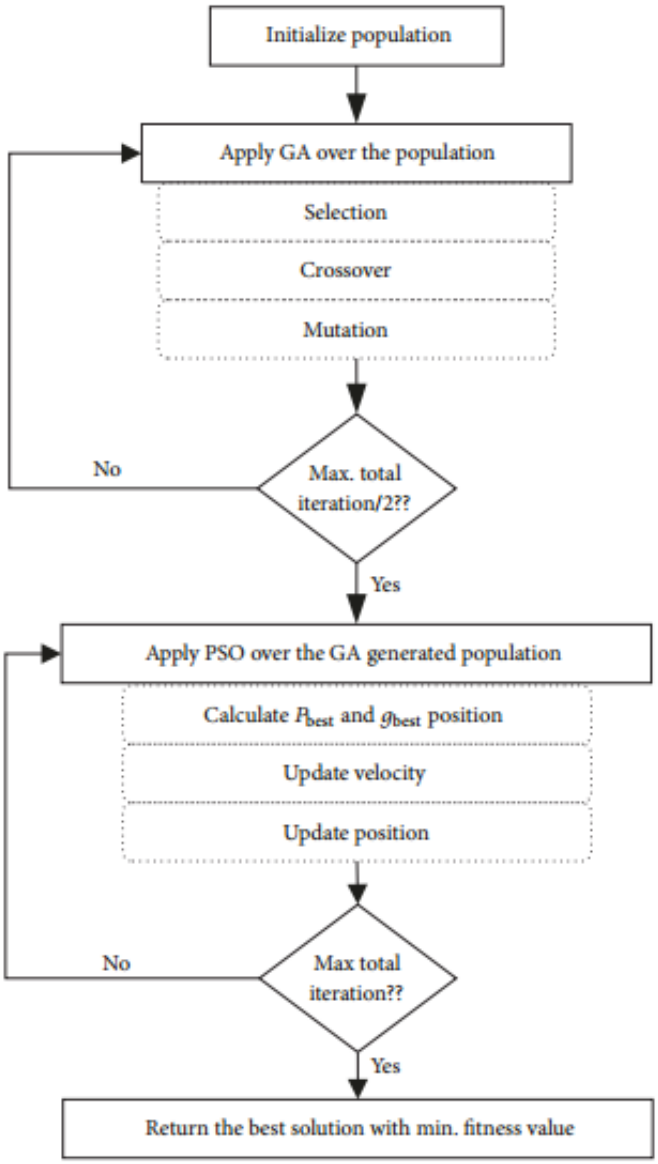

FIGURE 2: The flowchart of the GA-PSO algorithm [30].

GA-PSO算法从生成随机数量开始，并将特定数量的迭代定义为算法的参数。

算法1：锦标赛选择方法

```
Input: the chromosomes
Output: fitnesschromosome
Set the tournamentSize = n
   For i = 0 to tournamentSize
      id = Math.random( )* chromosome.size( )    // select chromosome randomly
      tournament[i] = get_chromosome (id)
   End For
fitness ← tournament.getFitnest( ) //return the fitness value in tournament group.
```

ALGORITHM 1: The tournament selection method.

GA-PSO算法的流程图

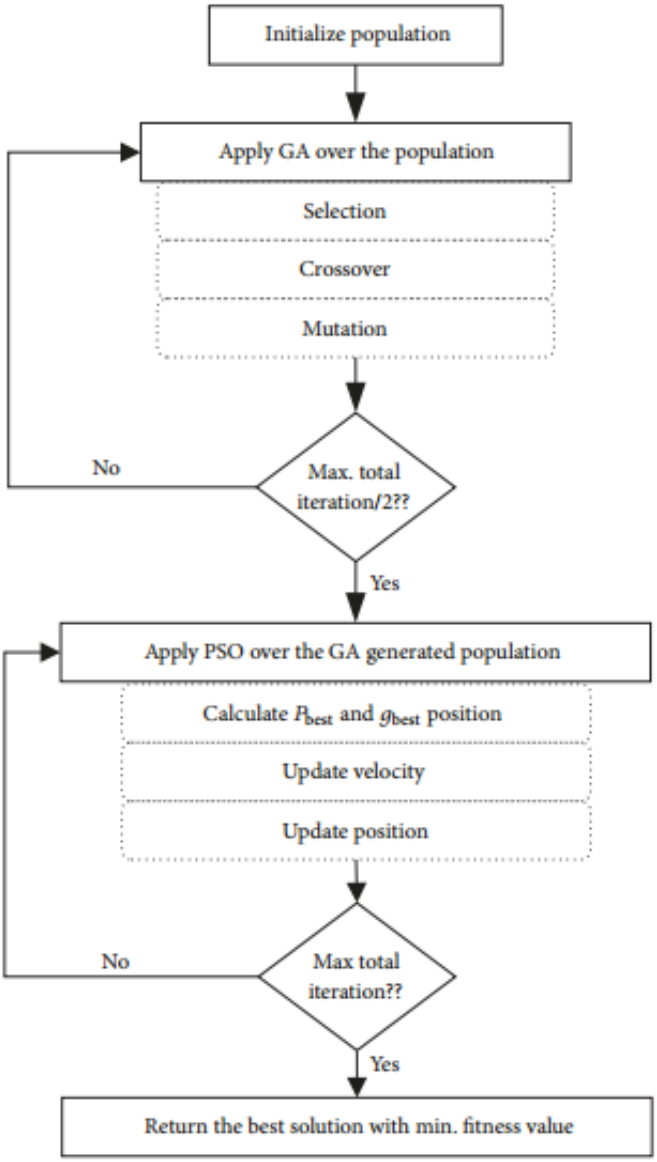

FIGURE 2: The flowchart of the GA-PSO algorithm [30].

算法2：交叉方法

```
Input: two chromosomes
Output: offspring chromosome
r = (Math.random( ) * chromosome.length)
    For i = 0, j = 0 to r
      offspring _chromosome[ j] = chromosome1[i]
    End For
    For i = r to chromosome.length
      offspring_chromosome[ j] = chromosome2[r]
    End For
```

ALGORITHM 2: The crossover method.

算法3：变异方法

```
Input: offspring_chromosome          //returned from crossover operator
Output: Newchromosome
Set mutationRate = 0.5
    If (Math.random( ) ≤ mutationRate)
        t1 = Rand [0, 1] * offspring_chromosome.length    // select a random number t1
        t2 = Rand [0, 1] * offspring_chromosome.length    // select a random number t2
            If offspring_chromosome [t1] !=offspring_chromosome[t2]
              Swap (offspring_chromosome [t1], offspring_chromosome[t2])
            End If
    End If
```

ALGORITHM 3: The mutation method.

算法4：演化(gbest)和(pbest)值

```
Input: particles
Output: (gbest) and (pbest) vlues
Set pbest = null; gbest = null; k=0;          //k is the index of the particles.
 While not Reach max particles.size do
        If pbest[k] == null ‖ pbest[k].getFitness() > particles[k].getFitness()
            pbest[k] =particles[k];
        End If
        If gbest == null ‖ pbest [k].getFitness() < gbest.getFitness()
          gbest = particle(pbest[k]);
          k = k + 1;
        End If
 Repeat    // until the last particle
```

ALGORITHM 4: Evolve (g_{best})and (p_{best}) values.

算法5：更新速度矩阵

```
Input: velocity values
Output: updated velocity values
Set k = 0; c1 = 1; c2 = 1.1; r1, r2 = rand[0, 1];    //k is the index of the particles individuals
    While not reach max particles.length do
        If Particle[k] == pbest[k]
          velocity[k] -= c1 * r1;          //r1, r2 are random numbers
          Else
          velocity[k] += c1 * r1;          //c1, c2 are acceleration coefficient
        End If
        If Particle[k] == gbest[k]
          velocity[k] -= c2 * r2;
          Else
          velocity[k] += c2 * r2;
        End If
    k = k +1;
  Repeat                                   //until the last particle
```

ALGORITHM 5: Update the velocity matrix.

算法6：更新位置矩阵

```
Input: updated velocity values
Output: updated particles position
Set j = 0;                                   // j is the index of the particle individuals
While not reach max particles.size ( ) do
        Maxvelocity_1 ← get_max_1(Particle_j, velocity values)
        Maxvelocity_2 ← get_max_2(Particle_j, velocity values)
        Swap(Particle_j[Maxvelocity_1], Particle_j[Maxvelocity_2])
           j = j + 1;
Repeat          //until the last particle
```

ALGORITHM 6: Update the position matrix.

算法7：提出的算法

```
Input: workflow W {N, E} and set of resources {VM_1, VM_2, VM_3, ..., VM_j}
Output: g_best                                   // the best solution to allocate W over VM_j
  For i = 0 to p
    population ← randomize() // initialize population, P is the population size
  End For
  While not Reach n/2 do                         // n is number of iterations
        While not Reach max P do
        chromosom_j ← tournament(population)    //selection operator
        chromosome_i ← tournament(population)
        offspring_chromosome_j ← crossover(chromosome_j, chromosome_i)
        Newchromosome_j ← mutation(offspring_chromosome_j)
        Repeat
  Repeat
Set Newchromosome_j as particle_j              // j is the index of the particles
Initialize particles position and velocity randomly
Calculate the (g_best) and (p_best) values
  While not Reach n do
  velocity matrix ← update(particle_j velocity)
  position matrix ← update(particle_j position)
  Repeat
```


ALGORITHM 7: The proposed algorithm.

创新贡献

针对云环境下的工作流任务调度问题，提出并实现了一种基于Workflowsim模拟器的GA-PSO算法。并与已知的遗传算法、PSO、HSGA、WSGA、MTCT等算法进行了性能比较。该算法的目的是在考虑工作流任务执行顺序的情况下，保证可用虚拟机之间的工作负载的公平分配，以减少云计算环境中工作流应用程序的最大使用时间和处理成本。

效果评价

为了评估所提出的算法，所提出的GA-PSO算法使用WorkflowSim来实现。 WorkflowSim通过提供适用于应用不同调度算法的环境，提供更高层的工作流管理，从而扩展了现有的CloudSim仿真器。此外，为了评估所提出的GA-PSO算法的性能，将所提出的GA-PSO算法的结果与现有的工作调度算法进行了比较。

执行实验的结果

Table 5: The result of the executed experiments.

Algorithm	Makespan (sec)	Execution cost ($)	Load balance (rate)
Scenario One			
GA-PSO	95.09	16.85	9.76
GA	197.65	52.68	52.58
PSO	101.21	18.16	21.33
Scenario Two			
GA-PSO	116.01	49.89	13.81
GA	250.89	86.34	61.93
PSO	155.31	62.86	18.23
Scenario Three			
GA-PSO	233.78	127.74	33.03
GA	345.72	137.09	49.2
PSO	253.44	133.55	41.82
Scenario Four			
GA-PSO	1585.6	1021.42	73.83
GA	2402.28	1529.23	134.67
PSO	1802.31	1200.41	90.15

最大完工时间的平均结果,执行成本,不同的负载平衡算法

Table 6: Average results in makespan, execution cost, and load balance for the different algorithms.

Methods	Avg. makespan	Avg. execution cost	Avg. load balance
Hybrid GA-PSO	507.62	303.975	32.6075
GA	799.135	451.335	74.595
PSO	578.0675	353.745	42.8825

不同数量任务的平均完工时间比较

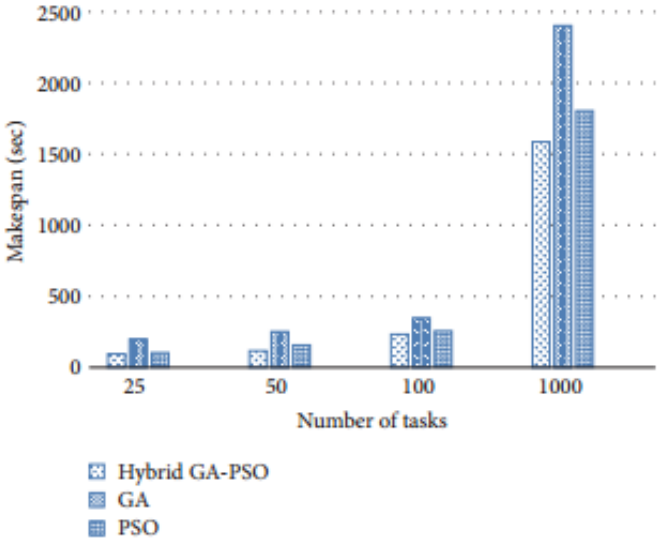

FIGURE 5: Comparison of the average makespan over different number tasks.

执行的算法的运行时间

TABLE 7: The running time of the executed algorithms in seconds.

Method/number of Tasks	25 tasks	50 tasks	100 tasks	1000 tasks	2000 tasks	3000 tasks
GA	0.869465	0.888796	1.093582	21.321738	23.4338	28.28996
PSO	0.761534	0.871797	1.037802	18.515041	18.65318	23.99583
Hybrid GA-PSO	0.764333	0.873796	1.025266	17.576722	18.00189	22.44825

不同数量任务的平均执行成本比较和平均负载平衡率的比较

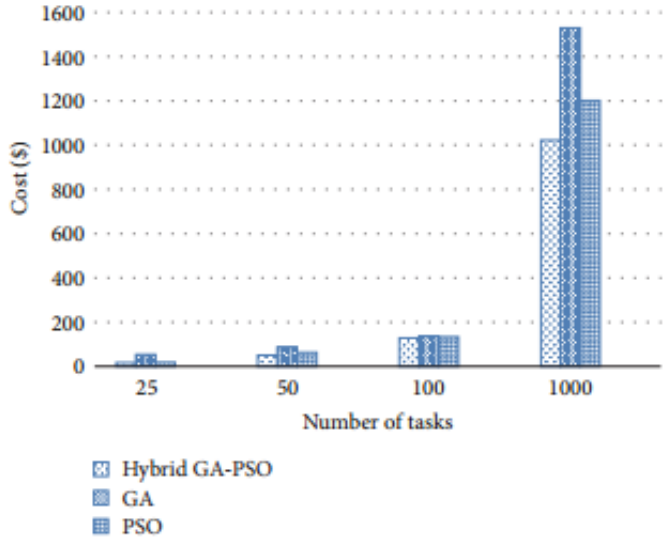

FIGURE 6: Comparison of the average execution cost over different number tasks.

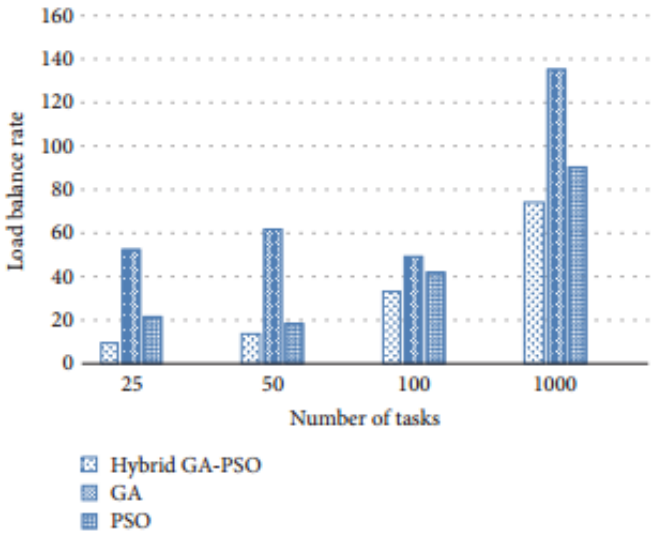

Figure 7: Comparison of the average load balancing rate over different number of tasks.

GA-PSO与HSGA实验结果与分析

Table 8: GA-PSO versus HSGA simulation parameters.

Parameter	Value
Number of tasks in application	20–100
Task lengths	12–72 ($\times 10^5$ MI)
Number of resources	30
Resource speeds	500–1000 (MIPS)
Bandwidth between resources	10–100 (mbps)

Table 9: GA-PSO versus HSGA experiment results.

Methods	Avg. makespan	Avg. load balance
GA-PSO	28191.96	2.23
HSGA	35000	2.63

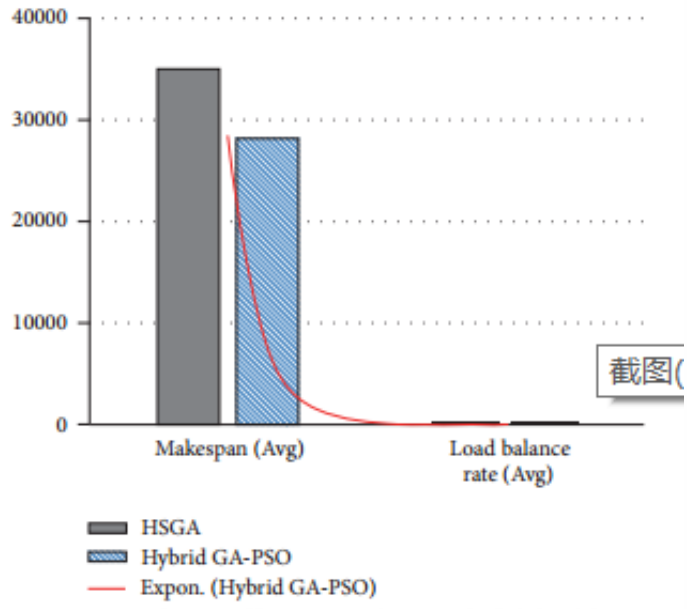

Figure 8: GA-PSO versus HSGA analysis of results.

GA-PSO与WSGA实验结果与分析

Table 10: GA-PSO versus WSGA simulation parameters.

Parameter	Value
Population size	20
Selection method	Roulette wheel
Crossover method	Single point crossover
Mutation rate	0.1
The number of resources	3–14
Number of tasks in application	50–100
Number of iterations	200

Table 11: GA-PSO versus WSGA experiment results.

Methods	Avg. makespan	Avg. execution cost
Hybrid GA-PSO	84.875	5.195
WSGA	93	7.695

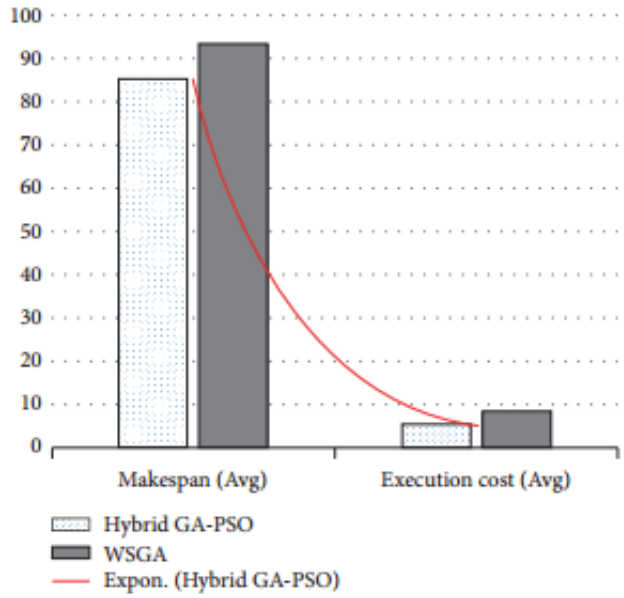

Figure 9: GA-PSO versus HSGA analysis of results.

GA-PSO与MTCT实验结果与分析

Table 12: GA-PSO versus MTCT simulation parameters.

Parameter	Value
The number of resources	20
Resource speeds	500–1000 (MIPS)
Bandwidth between resources	20 (mbps)

Table 14: GA-PSO versus MTCT experiment results.

Methods	The makespan (sec)	The execution cost ($)
Montage		
Hybrid GA-PSO	1.12	1.04
MTCT	1.4	1.4075
CyberShake		
Hybrid GA-PSO	0.9875	1.12
MTCT	1.365	1.3725
Epigenomics		
Hybrid GA-PSO	1.23	1.112
MTCT	1.3525	1.36
LIGO		
Hybrid GA-PSO	1.1075	1.132
MTCT	1.4975	1.4225

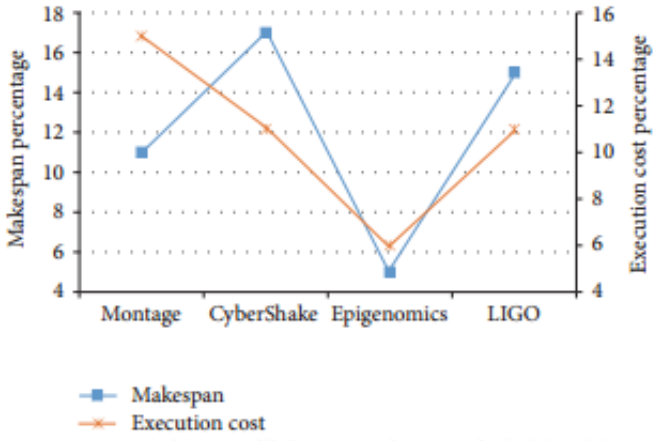

Figure 10: GA-PSO versus MTCT analysis of results.

个人感想

本文的优势

本文提出了一种混合遗传算法-PSO，可以有效地将任务分配给资源。混合遗传算法PSO的目标是在云计算环境中，降低异构资源上依赖任务的生成时间和成本，平衡依赖任务的负载。实验结果表明，与GA、PSO、HSGA、WSGA和MTCT算法相比，GA-PSO算法可以缩短工作流任务的总执行时间。此外，它还降低了执行成本。此外，它还提高了工作流应用程序对可用资源的负载平衡。最后，与其他算法相比，该算法收敛速度更快，收敛质量更好。

本文的劣势

工作可以扩展到异构环境中的多个数据中心。此外，Workflow应用程序的分布可以扩展到两个级别：当Workflow任务到达Service Broker时，以及当Workflow任务根据任务的大小和每个VM的速度分布到每个DC的可用VM时。可以通过实时云环境验证证明。此外，可以通过使用动态工作流来改进工作，这样用户在运行时可以更灵活地更改工作流任务的特性。