文章编号:1671 - 6833(2014)02 - 0055 - 05

云环境下基于化学反应优化的两阶段能耗感知任务调度策略

毋 琳, 王玉璟, 阎朝坤

(河南大学 计算机与信息工程学院,河南 开封 475004)

摘 要: 随着云数据中心能耗问题的日益严峻,过去以时间优化为目标的 BoT(Bag of Tasks)调度算法无法适应于当今的云计算环境. 为了在保障用户执行体验的同时,有效降低数据中心的能耗开销,提出了一种两阶段能耗感知任务调度算法 DH-CRO 算法,以 Makespan 作为主要优化目标,在此基础上基于动态电压频率调整技术调节能耗指标. 实验表明,DH-CRO 算法与遗传算法、Min-Min 算法、化学反应优化算法相比,能够更有效地优化完成时间,与此同时还能在一定程度上降低系统的总能耗,可以作为云环境的任务调度策略.

关键词: 云计算;任务调度;能量感知;CRO 算法;完成时间

中图分类号: TP393 **文献标志码**: A doi:10.3969/j.issn.1671 - 6833.2014.02.013

0 引言

基于虚拟化技术的云计算模式以其弹性扩展、按需付费、资源共享等特点受到了产业界和学术界的关注,已经成为科学计算、在线数据存储以及各类网络应用的重要基础设施. 作为生物信息、天体物理等科学领域的一种典型应用类型,BoT(Bag of Tasks)[1]任务的执行环境已经逐渐由传统的高性能集群、网格平台向新型的云计算环境转变,促使一些学者和机构开始研究如何有效执行 BoT 任务的问题.

早期针对 BoT 任务的调度研究主要集中于优化执行时间(Makespan),如 Min-Min 算法、Max-Min 算法、Sufferage 算法、遗传算法等,属于性能驱动的任务调度[2]. 随着效用计算的提出,一些研究将 BoT 调度目标扩展到执行费用,提出了一些针对时间及费用双目标优化的启发式调度算法. 文献[3]中,将粒子群算法和蚁群算法相结合,提出了一种云环境下的任务调度算法,从任务执行时间、执行费用和资源可靠性等 QoS 因素方面和虚拟资源的负载均衡等方面都进行了改进. 文献[4]采用 Min-Min、Max-Min 和 Sufferage 算法等多种经典调度算法,对云环境下系统吞吐量、调度收益、批调度完成时间和负载均衡指数等多目标进行优化.

然而,这些算法主要是从提高用户执行体验的角度出发,较少考虑资源提供商的利益,特别是基于大规模数据中心的云服务提供商因执行 BoT 任务导致的能耗花费.

目前,能耗感知的任务调度算法被更多的研究者所关注. 针对能耗感知的 BoT 任务调度问题,文献[5]基于遗传算法给出了调度策略. 然而,遗传算法存在搜索效率低、个体多样性差异及早熟现象的问题. 文献[6]针对高性能计算系统中的并行任务调度,提出了具有优先限制的双目标优化方法,对调度性能提高的同时,通过动态电压频率调整(DVS)技术控制系统能耗. 文献[7]针对云环境中的并行任务调度,采用基于遗传算法的方法,实现在保证并行任务截止时间的条件下,有效地降低系统的能耗开销.

化学反应优化算法[8]是 2010 年 Albert Y. S. Lam 等提出的一种元启发式算法,具有遗传算法和模拟退火算法的优点且有更加灵活的结构. 笔者将化学反应优化算法应用到能耗感知的 BoT 调度问题中,并对其反应操作进行了改进,提出了一种两阶段能耗感知任务调度算法 DH-CRO 算法,该算法可以在优化应用完成时间,保障用户执行体验的同时,最大程度地降低应用执行的能耗

收稿日期:2013 - 11 - 24;**修订日期**:2014 - 01 - 14

基金项目:河南省科学技术重点研究资助项目(14A520042)

作者简介:毋琳(1978 -),女,河南焦作人,河南大学讲师,硕士,主要从事网格计算与云计算方向的研究,E - mail:henuwl@126.com.

开销.

1 任务调度模型

1.1 资源任务模型

假定采用二元组 C = {R,T} 表示资源任务模型,其中各元组具体含义如下

(1) $R = \{r_1, r_2, \cdots, r_m\}$ 表示 m 个计算资源的集合,r_i 表示第 i 个资源.

R 中所有资源均支持 DVS 技术. 资源 r_i 可以表示为一个二元素的向量 $r_i = <cc_i, s^i>$,其中 cc_i 表示资源 r_i 的计算能力,s^i 表示资源 r_i 的供应电压策略. 若策略 s^i 有 $l_{\max}$ 个 DVS 级别,$V_i = [(v_{s_0}(i), f_{s_0}(i)), \cdots, (v_{s_{l(max)}}(i), f_{s_{l(max)}}(i))]^T$ 表示策略 s^i 供应电压与频率的关系矩阵,其中 $v_{sl}(i)$ 表示供应电压,$f_{sl}(i)$ 表示工作频率,范围是[0,1].

(2) $T = \{t_1, t_2, \cdots, t_n\}$ 表示 n 个待处理的相互独立的任务集合,t_j 表示第 j 个任务.

1.2 完成时间计算

任务执行时间矩阵定义为 $ETC = [ETC[i,j]]_{m \times n}$,其中 $ETC[i,j]$ 表示任务 t_j 在资源 r_i 上的执行完成时间. 根据资源任务模型,$ETC[i,j]$ 的计算公式可表示为

$$ETC[i,j] = \left[\frac{1}{f_{s_0}(i)} \times ETC[i,j], \cdots, \frac{1}{f_{s_{l(max)}}(i)} \times ETC[i,j]\right]. \tag{1}$$

1.3 能耗计算

资源 r_i 在执行所分配任务时的平均能耗为

$$E_i = \sum_{\substack{j \in T(i) \\ l \in L(j)}} \gamma \cdot f \cdot ETC[i,j] \cdot [(v_{s_l}(i))_j]^2 + \gamma \cdot f_{s_{\min}}(i) \cdot [v_{s_{\min}}(i)]^2 \cdot Idle_i, \tag{2}$$

式中:$T(i)$ 为分配给资源 r_i 的任务集合;$L(j)$ 为资源 r_i 行任务时 DVS 级别的集合;γ 对于给定设备来说是一个常数;$v_{s\min}(i)$、$f_{s\min}(i)$ 分别表示空闲时间资源 r_i 转换至休眠状态时的电压和频率;$Idle_i$ 表示资源 r_i 的空闲时间.

云计算环境下任务调度模型是寻找最优的调度方案,使任务完成时间最小的同时,系统总能耗最低. 该调度模型可以形式化描述为

$$\text{Min energy} = \sum_{i \in R(i)} E_i$$

$$\text{s.t.} \sum_{l \in L_i} \left[\frac{1}{f_{s_l}(i)} \cdot ETC[i,j]\right] \leqslant \text{time}; \forall i \in R(i), \tag{3}$$

式中:time 值应取资源 R 中完成时间的最大值,计算公式可表示如下

$$\text{time} = \min \max_{i \in R(i)} \sum_{j \in T(i)} ETC[i,j]. \tag{4}$$

2 DH-CRO 算法

2.1 CRO 基本原理

CRO 算法是一种基于种群的元启发式优化算法. 算法中的主体是分子,分子通过分子结构、势能 PE、动能 KE、碰撞次数等属性来描述. 在化学反应中,每个分子代表所要求解问题中的一个解,分子的势能 PE 对应求解问题的目标函数值,分子的动能 KE 可以使得分子进入一个高势能状态,避免解陷入局部最优. CRO 算法遵循能量守恒原则,所有化学反应都在一个密闭容器内完成,分子和能量缓冲池 buffer 总能量在整个迭代反应过程中保持不变.

化学反应算法中,具有能量的分子会发生移动和碰撞,主要涉及 4 种基本反应操作,即器壁碰撞反应、分解反应、分子间轻微碰撞反应以及合成反应.

2.2 分子操作的设计

2.2.1 器壁碰撞反应

器壁碰撞反应是单个分子与容器壁碰撞的轻微反应,该反应使发生碰撞的分子结构发生一定变化,相当于在该分子的邻域解空间内进行局部搜索.

笔者所描述的 CRO 改进算法中,将爬山算法引入到器壁碰撞反应,利用爬山算法的贪心特性,使该反应从局部解空间中得到一个最优解.

但由于 CRO 算法是以任务为中心的调度结构,这与爬山算法对数据结构的要求不符,故在应用爬山算法进行局部优化时需要先转换调度结构. 图 1 为爬山算法的 hash 表结构图.

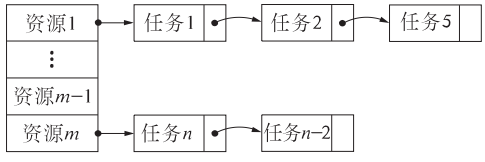

图 1 爬山算法的数据结构示意图

Fig. 1 Data structure of hill chimbing algorithm

改进后的器壁碰撞反应算法流程如下

If OnwallIneffectiveCollision then

Transform scheduling structure

For i = 1 to 5

Enforced HC algorithm to optimize the reaction

Update Scheduling structure of CRO and HC

Endfor

Endif.

2.2.2　分解反应

分解反应是分子与墙壁碰撞后的分解. 这种反应较器壁碰撞反应而言,需要较高的能量,此时会动用 buffer 中的能量促进分解反应的进行.

本实验中,分解反应定义为将分子结构序列移动随机位数的方式来进行. 例如,某解分子的初始序列为 S:[1,2,3,4,5,6],在某次分解操作后得到新的解分子 S1:[2,3,4,5,6,1]和 S2:[5,6,1,2,3,4].

2.2.3　分子间轻微碰撞反应

分子间轻微碰撞反应中分子与容器壁没有接触,因此不会动用 buffer 中的能量,只是 2 个分子之间的能量交换. 这个反应相当于是在解空间内的局部搜索邻居解.

本实验分子间轻微碰撞反应采取单点交叉的方式. 例如,两个解分子 S1:[2,5,3,1,4,6]和 S2:[5,6,3,2,4,1],碰撞后新分子为 S1:[2,5,3,2,4,1]、S2:[5,6,3,1,4,6].

2.2.4　合成反应

合成反应是 2 个分子互相剧烈碰撞,最终结合成一个分子的反应操作. 一般将该反应会通过简单的交叉,变异等操作来完成,这样得到的解分子仍然有些盲目. 如果在合成反应中加入差异演化算法,会使演化在单纯跳出局部最小值的同时保证更强的多样性.

差异演化算法的变异反应是对当前种群 G 选取 3 个互不相同的个体 $X_{r1,G}$、$X_{r2,G}$、$X_{r3,G}$,这些个体根据式(5)运算形成下一代新个体 $V_{i,G+1}$. 差异演化包含 2 个可调参数:缩放因子 F 和交叉概率 CR. 缩放因子用于差向量,交叉概率制定初始向量与目标向量发生交叉的概率

$$V_{i,G+1} = X_{r_1,G} + F \times (X_{r_2,G} - X_{r_3,G}). \qquad (5)$$

笔者的 DH-CRO 算法,将差异演化算法引入到合成反应中,利用差异演化算法的变异、重组和选择等操作,在更大范围搜索最优解分子.

改进后的合成反应算法流程如下

If　Synthesis　then

Randomly selected three different solutions from the solution space

Enforced DE algorithm, Get the new molecules

Update solution space and buffer

Endif.

综上所述,DH-CRO 算法的整体流程如图 2 所示. 算法主要包括 3 个阶段:第一阶段为初始化,随机生成解,形成均匀分配的解空间,增加全局搜索能力;第二阶段为迭代求解期,解空间内的分子其状态发生移动和碰撞,期间在器壁碰撞反应中使用爬山算法对解空间进行局部最优的搜索,在合成反应中采用差异演化算法生成新的解分子;第三阶段为最优解输出,当解分子趋于最低能量状态时,即为所求的最优解.

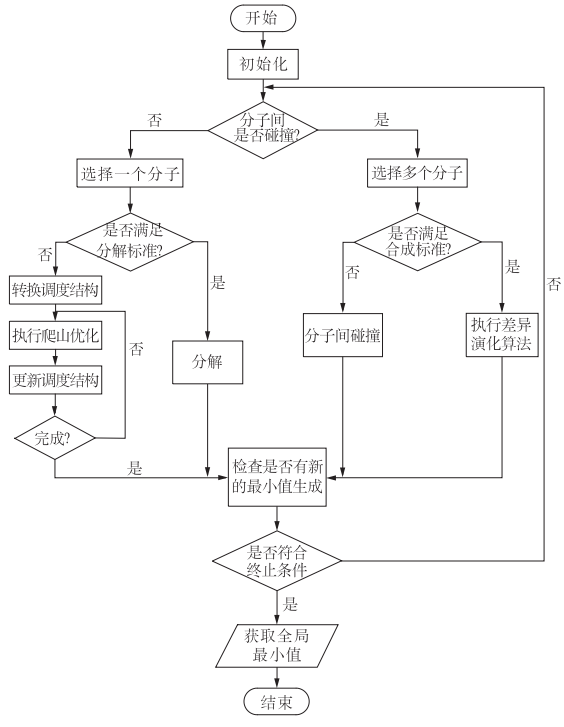

图 2　DH-CRO 算法的基本流程图

Fig. 2　Flow chat of DH-CRO algorithm

3　实验与分析

3.1　实验环境设置

为了验证 DH-CRO 算法的性能,笔者在 Intel Core i7 3.4GHz, 4GB DDR3 SDRAM 的 PC 机上,采用 MyEclipse10.0 的 Java 集成开发环境,实现了经典的 GA、Min-Min 算法和 CRO 算法,以及笔者提出的 DH-CRO 算法. 表 1 给出了两种实验场景,对以上 4 种算法分别进行了 200 次模拟实验,取实验结果的平均值作为最终的实验结果,在完成时间和系统总能耗方面的性能表现进行比较.

系统初始化时,设置资源计算能力为(1 000,175),任务负载为(250,43.75),DVS 类别按照表 2 随机生成.

表1 实验场景Ⅰ、Ⅱ设置表

Tab.1 Configuration of experiment scene Ⅰ and Ⅱ

场景Ⅰ(任务数为定值512)		场景Ⅱ(资源数为定值32)	
资源数	任务资源比	任务数	任务资源比
128	4	128	4
84	6	192	6
57	9	288	9
32	16	512	16
16	32	1 024	32

*注:任务资源比为任务数/资源数.

表2 DVS矩阵取值表

Tab.2 Value of DVS matrix

类别	第一类		第二类		第三类	
级别	电压/V	频率/MHz	电压/V	频率/MHz	电压/V	频率/MHz
0	1.5	1.0	2.2	1.0	1.75	1.0
1	1.4	0.9	1.9	0.85	1.4	0.8
2	1.3	0.8	1.6	0.65	1.2	0.6
3	1.2	0.7	1.3	0.5	1.0	0.4
4	1.1	0.6	1.0	0.35		
5	1.0	0.5				
6	0.9	0.4				

本实验中CRO算法参数设置采用了文献[8]中取值.差异演化算法的缩放因子F设为0.6,交叉概率CR设为0.9,算法中迭代次数为100.GA算法的实验参数设置参考文献[5].

3.2 实验分析

场景Ⅰ中,4种算法在任务调度实验中的结果如表3、4所示.

表3 场景Ⅰ中GA和Min-Min算法性能结果

Tab.3 Result contrast with algorithm of GA and min-Min in scene Ⅰ

资源数	GA		Min-Min	
	时间/s	能量/kw	时间/s	能量/kw
16	8.884 6	405.918	12.362 41	455.697
32	4.779 75	411.202 28	10.025 11	490.017 27
57	3.239 09	369.912 07	5.841 13	464.310 17
84	2.584 41	324.397 82	3.742 25	466.656 63
128	1.977 92	350.210 72	2.814 47	473.312 67

表4 场景Ⅰ中CRO和DH-CRO算法性能结果

Tab.4 Result contrast with algorithm of CRO and DH-CRO in scene Ⅰ

资源数	CRO		DH-CRO	
	时间/s	能量/kw	时间/s	能量/kw
16	9.947 05	393.138 05	7.879 11	378.463 01
32	7.255 83	386.637 46	4.104 59	431.874 37
57	3.609 28	368.009 33	2.370 33	381.882 09
84	2.942 38	360.862 76	1.673 04	414.368 02
128	2.176 56	360.520 01	1.213 35	386.038 3

从表3、4中可以看出,在任务数确定(任务数=512)时,随着资源数的增加,4种算法的完成时间都有所减小,其中笔者提出的DH-CRO算法表现最为突出.当资源数为32时,DH-CRO算法的完成时间为4.104 59 s,相较于GA算法、Min-Min算法和CRO算法分别提高了14.1%、59.5%、43.4%.当资源数达到128时,该比率分别为37.1%、56.7%和44.3%.以上数据说明,资源数的不断增加,使给定任务的可调度范围不断扩大,而DH-CRO算法对于全局最优解具有较强的搜素能力,更加快速地完成资源的最优调配,因此对于完成时间性能的优化明显.

由于任务数不变,系统所需要消耗的能量基本相当,因此从表中的能耗值来看,无论资源数如何变换,4种算法能耗都大体维持不变.

场景Ⅱ中,4种算法在任务调度实验中的结果如表5、6所示.

表5 场景Ⅱ中GA和Min-Min算法性能结果

Tab.5 Result contrast with algorithm of GA and min-min in scene Ⅱ

任务数	GA		Min-Min	
	时间/s	能量/kw	时间/s	能量/kw
128	1.584 97	84.844 14	2.074 43	116.594 33
192	2.153 06	135.711 43	4.558 14	181.980 33
288	3.161 28	209.020 44	6.214 8	250.395 33
512	4.779 75	381.412 54	7.996 34	464.168
1 024	9.913 04	868.170 41	12.332 81	994.298 67

表6 场景Ⅱ中CRO和DH-CRO算法性能结果

Tab.6 Result contrast with algorithm of CRO and DH-CRO in scene Ⅱ

任务数	CRO		DH-CRO	
	时间/s	能量/kw	时间/s	能量/kw
128	1.739 56	85.336 61	1.123 14	89.139 54
192	2.570 88	140.069 06	1.612 75	149.739 88
288	3.793 83	170.420 04	2.353 29	222.770 71
512	7.255 83	374.123 1	4.104 59	393.785 73
1 024	11.238 05	820.285 03	8.296 73	926.655 57

从表5、6中可以看出,在资源数确定(资源数=32)时,随着任务数的不断增加,4种算法的完成时间和能耗都呈上升趋势.在完成时间优化方面,仍然是笔者提出的DH-CRO算法表现最优.当任务数达到1 024时,DH-CRO算法的完成时间值为8.296 73 s,比GA算法、Min-Min算法、CRO算法的完成时间分别缩短16.3%、32.7%和26.2%.在能耗优化方面,CRO算法表现最优,

Min-Min 算法表现最差,笔者提出的 DH-CRO 算法略高于 GA 算法. 这是由于 Min-Min 算法是以任务执行时间为目标的,并未考虑能耗优化问题,因此能耗值最高. DH-CRO 算法是在 CRO 算法的基础上使用了爬山算法作为其反应操作,这样造成在最大程度上优化任务完成时间的同时,使用 DVFS 调节降低能耗可操作范围会变小. 对于本调度目标来说,在完成时间大幅度优化的前提下,有一定的能耗成本是可以容忍的.

综上所述,在不同场景中,采用 DH - CRO 算法作为云环境中的任务调度策略,可以在最大程度上优化应用执行完成时间,保障用户执行体验的同时,相应地降低应用执行的能耗开销,在一定程度上节省服务提供商的执行成本.

4　结论

利用爬山算法、差异演化算法对 CRO 算法的操作反应进行改进,提出了能够适用于云环境能耗感知任务调度的 DH-CRO 算法. 下一步工作中,需要对于 CRO 算法中各个反应的发生临界值也应当继续进行深入研究,设置不同的临界值,使得算法总体能够取各算法之长,大幅度的提升优化效果.

参考文献:

[1] RICCIARDI S, CAREGLIO D, SANTOS-BOADA G, et al. Saving energy in data center infrastructures [C]//Data Compression, Communications and Processing (CCP), 2011 First International Conference on. IEEE, 2011: 265 - 270.

[2] 罗慧敏,阎朝坤,罗军伟. 截止期约束下 QoS 导向的网格任务调度算法[J]. 河南大学学报:自然科学版,2010:40(6):632 - 636

[3] 王科. 基于改进蚁群算法的云计算任务调度策略研究[D]. 杭州:杭州电子科技大学计算机学院,2012.

[4] 房欢. 云计算中的任务调度及重调度优化决策问题的研究[D]. 成都:电子科技大学航空航天学院,2012.

[5] KOLODZIEJ J, KHAN S U, XHAFA F. Genetic algorithms for energy-aware scheduling in computational grids[C]//P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on. IEEE, 2011: 17 - 24.

[6] LEE Y C, ZOMAYA A Y. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling[C]//Cluster Computing and the Grid, 2009. CCGRID09. 9th IEEE/ACM International Symposium on. IEEE, 2009: 92 - 99.

[7] 曹洁,曾国荪. 云环境下计算资源动态能耗感知的并行任务调度方法[J]. 计算机科学,2013,40(10): 39 - 44.

[8] LAM A Y S, LI V O K. Chemical-reaction-inspired metaheuristic for optimization[J]. IEEE Transactions on Evolutionary Computation, 2010, 14(3): 381 - 399.

A Novel Two-step Task Scheduling Strategy Based on Chemical Reaction Optimization in Cloud Environment

WU Lin, WANG Yu-jing, YAN Chao-kun

(School of Computer and Information Engineering, Henan University, Kaifeng 475004, China)

Abstract: As the energy problem of cloud data center is becoming more and more serious, the BoT (Bag of Tasks) scheduling algorithm considering only timespan can not adapt to the cloud computing environment. In order to guarantee users to perform experience, and also effectively reduce the energy consumption, this paper presents a two-step energy-aware task scheduling algorithm, called DH-CRO algorithm. DH-CRO taking makespan as the main optimization goals, meanwhile adjusts energy consumption by DVFS technique. Experimental results verify DH-CRO algorithm is more efficient than other algorithms, such as GA, Min-Min, CRO and so on. DH-CRO using in cloud environment can optimize the makespan to the maximum extent, and reduce the total energy consumption at the same time.

Key words: cloud computing, task scheduling, energy-awareness, CRO algorithm, makespan