计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(7):173−176 [doi: 10.15888/j.cnki.csa.006408]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

一种提高集群调度性能的改进型蚁群算法①

刘素芹, 张　千, 王俊爽

(中国石油大学 (华东) 计算机与通信工程学院, 青岛 266580)
通讯作者: 刘素芹, E-mail: liusq@upc.edu.cn

摘　要: 蚁群算法 ACO 能较好地应用于集群调度, 但其传统的信息素更新方式带来了性能匹配和负载均衡等问题, 影响了集群调度的性能. 针对这些问题, 提出了改进型蚁群算法 IACO, 通过引入性能匹配因子和负载均衡因子更合理地调整信息素, 缩短了作业处理时间, 提高了 CPU 利用率, 从而有效地提高了集群调度性能.

关键词: 集群调度; ACO; 性能匹配因子; 负载均衡因子; IACO

引用格式: 刘素芹,张千,王俊爽.一种提高集群调度性能的改进型蚁群算法.计算机系统应用,2018,27(7):173–176. http://www.c-s-a.org.cn/1003-3254/6408.html

Improved Ant Colony Algorithm for Improving Performance of Cluster Scheduling

LIU Su-Qin, ZHANG Qian, WANG Jun-Shuang

(College of Computer &Communication Technology, China University of Petroleum, Qingdao 266580, China)

Abstract: The Ant Colony Algorithm (ACO) can be applied to cluster scheduling better, but its traditional pheromone update method brings the performance matching and load balancing and so on, which affects the performance of cluster scheduling. In order to solve these problems, an Improved ACO (IACO) is put forward. The pheromone is adjusted more reasonably by adjusting the performance matching factor and the load balancing factor. It is observed that the processing time is shortened, the CPU utilization is improved, and the performance of the cluster is improved effectively.

Key words: cluster scheduling; Ant Colony Algorithm (ACO); performance matching factor; load balancing factor; Improved ACO (IACO)

1　引言

集群资源进行调度成为影响集群性能的重要因素之一[1]. 文献传统的调度算法先来先服务算法 (First Come First Serve, FCFS) 等传统的调度算法难以满足大规模异构集群对资源调度的需求[2−4], 尤其是负载均衡问题更为突出[5,6], 蚁群算法 (Ant Colony Optimization, ACO) 等仿生型智能调度算法受到越来越多的关注.

国内外学者对蚁群算法在集群调度中的应用做了一定的研究[7,8], 认为其分布并行性、健壮性、可扩展性都比较适合于集群调度, 但是蚁群算法没有充分考虑资源与作业的匹配度和负载均衡问题, 导致集群的整体性能不能得以充分发挥, 所以需要对蚁群算法深入研究并加以改进, 以进一步提高集群调度的性能.

2　蚁群算法及在集群调度中的应用

2.1　蚁群算法原理

蚁群算法[8]是由意大利学者于 1991 年首先提出的一种具有群体智能的仿生优化算法, 借鉴生物界中蚂蚁在觅食过程中总能找到最短路径的思想. 蚁群算法用蚂蚁行走的路径轨迹求解问题的最优解, 蚂蚁的每

① 基金项目: 山东省自然科学基金 (Z2016FM017); 中央高校基本科研业务费专项基金 (16CX02046A)
Foundation item: Natural Science Foundation of Shandong Province (Z2016FM017); Special Fundation for the Fundamental Research Services for the Central Universities (16CX02046A)
收稿时间: 2017-10-20; 修改时间: 2017-11-10; 采用时间: 2017-11-27

© 中国科学院软件研究所 http://www.c-s-a.org.cn

条路径轨迹代表所求问题的一种解, 所有的蚂蚁都在解空间中独立地搜索, 当蚂蚁在行进过程中遇见尚未经过的路口时, 就从该路口延伸出的路径中随机地选择一条尚未走过的路径继续前进, 同时释放与该路径长度有关的信息素增强该路径的被选概率, 该路径长度越短, 释放的信息素就越多. 当后来的蚂蚁再次经过该路口时, 就选择信息素浓度高的路径, 同时释放更多的信息素, 形成正反馈机制. 同时引入信息素挥发机制, 避免残留信息素太多使启发信息失去参考价值的现象. 蚁群算法的这种自催化行为不需要外界提供其他的信息, 应用起来比较简单, 现已在旅行商问题、组合优化问题、二次分配问题等领域得到了良好的发展.

2.2 蚁群算法用于集群调度的策略

用蚁群算法解决集群调度问题的策略[9,10]为:

(1) 设组成蚁群的蚂蚁数量为 M, 每只蚂蚁均可独立求得一组作业调度方案.

(2) 集群系统初始化, 集群中所有节点都需提供自身的 CPU 个数及每个 CPU 的处理能力等参数, 为每个节点设置初始信息素.

(3) 异构集群系统中的管理节点在每个调度周期开始前就开始收集各节点的资源信息:

① 若有新节点加入集群, 就为这个新加入集群的节点根据它所提供的 CPU 个数及处理能力等参数设置初始信息素;

② 若集群中有节点因故障等原因掉线, 则将该掉线节点作停用标记;

③ 若集群中作停用标记的节点恢复, 就重新为该节点设置启用标志;

④ 集群中作业调度结束后, 依据作业的完成情况修改节点的信息素: 成功则奖励, 失败则惩罚.

(4) 蚂蚁个体根据概率公式随机选择一个可用的计算节点作为第一个作业分配的节点, 并且依据信息素计算出转移概率的大小, 按顺序决定下一个作业分配的节点. 重复此步骤, 直到所有的作业分配完成.

(5) 调度策略的优化目标是所有任务的总体执行代价最小和系统资源的负载平衡性最优.

当一个蚂蚁找到一种调度方案后, 就计算该调度方案的执行时间和负载均衡性. 当所有蚂蚁都找到调度方案后, 比较每个调度方案的执行时间和负载均衡性, 并以此修改各相关路径的信息素. 选择任务完成时间最短和负载均衡性最优的一组资源为最优方案.

2.3 信息素更新带来的问题

在蚁群算法中, 信息素的更新非常重要, 集群系统直接根据信息素更新后的大小决定下一个作业选择该集群节点的概率. 蚁群算法的信息素更新公式如式 (1) 和式 (2):

$$\tau_j^{\text{new}} = (1-\rho)\cdot\tau_j^{\text{old}} + \Delta\tau_j \tag{1}$$

$$\Delta\tau_j = \begin{cases} C_e \times k, & \text{作业成功完成}, C_e \text{为奖励因子} \\ C_p \times k, & \text{作业完成失败}, C_p \text{为惩罚因子} \end{cases} \tag{2}$$

其中, τ_j^{new}是集群节点当前的信息素, $(1-\rho)\cdot\tau_j^{\text{old}}$是残留的信息素, C_e 是奖励因子示, C_p 是惩罚因子, ρ 为信息素挥发系数, k 为作业所需计算能力.

从公式 (1) 和公式 (2) 可以看到, 信息素的更新方式有时会带来以下两方面的问题:

(1) 作业需求与集群节点资源性能不匹配

从公式 (2) 中蚂蚁根据作业完成情况释放信息素可以看出, 作业成功完成, 就对该集群节点进行奖励; 作业完成失败, 就对该集群节点进行惩罚. 但这种情况并没有考虑作业需求和集群节点资源性能的匹配问题, 很可能会出现小作业在大资源上执行的这种“大炮打蚊子”式的性能不匹配的问题.

(2) 集群各节点之间负载不均衡

公式 (1) 中信息素挥发系数 ρ 通常是固定的常数, 也即各集群节点上的信息素挥发程度相同. 但集群各节点的性能不同, 作业又是随机到达的, 系统运行一段时间后, 很可能会出现集群各节点之间负载不均衡的情况.

3 对蚁群算法的改进

针对信息素更新中存在的问题对蚁群算法进行改进, 称之为该进型蚁群算法 (Improved ACO, IACO) 算法, 这种改进包括两个方面: 一方面引入性能匹配因子提高性能匹配度, 把这种算法称为 PM-ACO (Performance Matching-ACO); 另一方面引入负载均衡因子优化负载均衡, 把这种算法称为 LB-ACO (Load Balance-ACO).

3.1 PM-ACO 算法

为了使得分配的资源和作业的需求更好的匹配, 达到物尽其用, 从而更好的发挥集群的整体性能, 进行资源调度时, 根据作业的需求计算每个作业到各个集群节点的匹配程度, 引入性能匹配因子 λ_{ij}. λ_{ij} 即用户作

© 中国科学院软件研究所 http://www.c-s-a.org.cn

业 i 所需资源与集群空闲节点 j 拥有资源的匹配度.

$$\lambda_{ij}=\sqrt{(realExetime-estimatedTime)^2} \tag{3}$$

其中, $realExetime$ 表示作业的实际运行时间, $estimatedTime$ 表示作业的预期运行时间, 作业对资源的需求与集群节点所拥有资源越接近, 性能匹配因子 λ_{ij} 也就越小.

引入性能匹配因子后, 蚂蚁根据作业完成情况释放信息素的公式 (4) 和公式 (5) 如下:

$$\lambda_{ij}:\begin{cases}<\lambda_0, & c\text{取大于1的奖励因子}\\ >\lambda_0, & c\text{取小于1的惩罚因子}\end{cases} \tag{4}$$

$$\Delta\tau_j=\Delta\tau_j+ck \tag{5}$$

其中, λ_{ij} 为性能匹配因子, λ_0 为匹配度阈值, $\Delta\tau_j$ 为信息素增量, c 为奖励因子或惩罚因子, k 为所需计算资源.

当作业在资源节点上执行完成后, 计算作业与该集群节点的性能匹配因子 λ_{ij} 的值, 并把 λ_{ij} 的值与匹配度阈值作比较. 如果 λ_{ij} 的值小于匹配度阈值, 信息素调节因子就取奖励因子, 增大该资源节点的信息素浓度, 使得此资源节点被选中的概率增大. 如果 λ_{ij} 的值大于匹配度阈值, 信息素调节因子就取惩罚因子, 减小资源节点的信息素浓度, 使得此资源节点被选中的概率降低. 这样, 对于资源处理能力需求小的作业就可能选择更匹配的处理能力低的集群节点. 队列后面对资源处理能力需求大的作业也就会到处理能力高的集群节点上去执行, 从而较好地解决了资源与作业不匹配的问题.

3.2 LB-ACO 算法

在蚁群算法中, 挥发系数的大小直接影响着算法的收敛速度. 挥发系数较小时, 计算节点的信息素较大, 算法收敛的也较快. 为了满足蚁群算法作业调度中负载均衡的要求, 系统需要不断的检测计算节点的负载及其作业完成情况. 为了保证负载均衡, 我们需要把各计算节点负载完成率的差值控制在一个较小的阈值之内. 为了动态的调整蚁群算法的挥发系数, 提高负载均衡性能, 引入负载均衡因子, 表达如公式 (6) 所示:

$$\omega_i=\frac{T_f}{T_a},\quad i=1,2,\cdots,m \tag{6}$$

其中, T_a 表示计算节点成功完成的任务数, T_f 表示该计算节点已经接收的所有任务总数. T_f 一定时, T_a 越大, 计算节点成功完成的任务数越多, ω_i 也越小, 表明计算节点的作业完成率越高; T_a 越小, 计算节点成功完成的任务数越少, ω_i 也就越大, 表明计算节点的作业完成率越低.

引入负载均衡因子后, 集群节点残留信息素改变如公式 (7) 所示:

$$\omega_i:\begin{cases}<\omega_0, & \text{减小信息素挥发系数}\rho\\ >\omega_0, & \text{增大信息素挥发系数}\rho\end{cases} \tag{7}$$

其中, ω_0 为负载均衡因子的初始阈值.

在为作业选择集群节点时, 需要计算该集群节点的负载均衡因子 ω_i 的值, 并将 ω_i 的值与初始阈值 ω_0 作比较. 若 ω_i 小于初始阈值时, 应该减小挥发系数, 增大集群节点的信息素值, 使该集群节点被选概率增大, 作业分配收敛于此资源节点; 反之, 若 ω_i 大于初始阈值时, 应该增大挥发系数, 减小计算节点的信息素值, 使该集群节点被选概率降低, 使算法的全局搜索能力增强, 以便找到更加匹配的资源.

3.3 IACO 用于集群调度的实现方案

改进后的蚁群算法应用于异构集群系统进行资源调度的实现方案如图 1 所示.

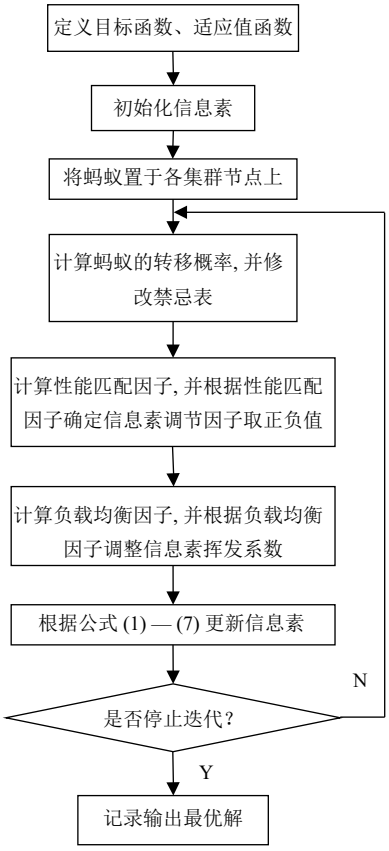

图 1 IACO 用于集群调度的实现方案

© 中国科学院软件研究所 http://www.c-s-a.org.cn

调度开始前, 异构集群系统中的管理节点就开始收集各节点的资源信息素信息. 若有新节点加入, 则对该节点进行初始化. 集群系统对每组作业调度时, 都需要创建一个新的蚁群. 蚁群中的每个蚂蚁找到一种调度方案后, 就计算该调度方案的执行时间和负载均衡性. 当所有蚂蚁都找到调度方案后, 比较每个调度方案的执行时间和负载均衡性, 并以此修改各相关路径的信息素. 选择任务完成时间最短和负载均衡性最高的一组资源为最优方案.

4 实验结果及分析

为了对改进的蚁群算法在集群调度中的应用效果进行测试, 在 100 个节点的异构集群上对真实的地震资料进行处理, 调度算法分别采用先来先服务算法 FCFS、蚁群算法 ACO、引入性能匹配因子的 PM-ACO 算法、引入负载均衡因子的 LB-ACO 算法和最终改进的蚁群算法 IACO, 主要对处理时间和 CPU 平均利用率进行对比, 实验结果如图 2 和图 3 所示.

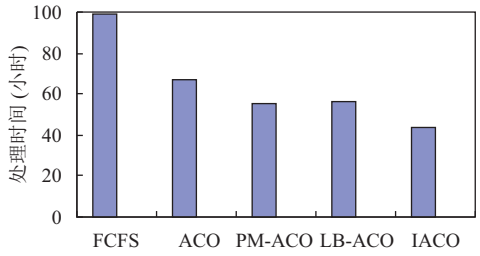

图 2 处理时间比较

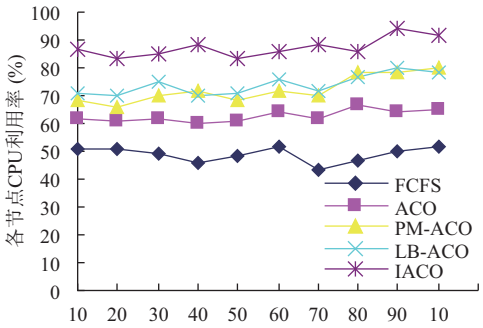

图 3 各节点 CPU 平均利用率对比

从实验结果可以看出, 蚁群算法 ACO 考虑到资源和作业的实际情况, 比传统的先来先服务算法 FCFS 有明显的优势; 引入性能匹配因子后的 PM-ACO 算法解决了“大炮打蚊子”的问题, 使得性能有所改进, 但负载均衡问题没有解决; LB-ACO 算法考虑了负载均衡问题, 但还有可能存在“大炮打蚊子”现象; IACO 较好地解决了这两方面的问题, 不论是处理时间还是 CPU 利用率均有明显提高.

由此可见, 传统的蚁群算法应用于集群调度时确实存在资源匹配和负载均衡的问题, 在这两个方面进行改进后能够较好地提高集群的调度性能.

5 结论

传统调度算法已经难以满足大规模异构集群的作业的调度, 蚁群算法比较适合于集群中的作业调度, 但是, 传统的蚁群算法在作业匹配和负载均衡方面还存在问题, 改进的蚁群算法 IACO 较好地解决了这一问题, 有效地提高了集群调度的性能.

参考文献

1 孙俊峰. 计算机集群性能瓶颈的研究与改进. 信息通信, 2015, (10): 105–107. [doi: 10.3969/j.issn.1673-1131.2015.10.061]
2 李永峰, 周敏奇, 胡华梁. 集群资源统一管理和调度技术综述. 华东师范大学学报 (自然科学版), 2014, (5): 17–30.
3 王文豪, 严云洋, 周静波. 基于负载均衡的 Min-Min 任务调度算法优化. 南京理工大学学报, 2015, 39(4): 398–404.
4 孙震宇, 石京燕, 姜晓巍, 等. 大型高能物理计算集群资源管理方法的评测. 计算机科学, 2017, 44(10): 85–90. [doi: 10.11896/j.issn.1002-137X.2017.10.016]
5 张腊, 刘淑芬, 韩璐. 基于负载均衡的任务调度算法. 吉林大学学报 (理学版), 2014, 52(4): 769–772.
6 张宝祥, 何利力. 高并发集群系统下的负载均衡技术研究. 工业控制计算机, 2017, 30(10): 76–77, 138. [doi: 10.3969/j.issn.1001-182X.2017.10.032]
7 冯焕霞, 刘莉, 李正淳. 异构集群下的动态任务调度策略. 软件导刊, 2014, 13(6): 23–26.
8 张风荣. 基于蚁群算法的 Hadoop 调度算法研究. 电脑与信息技术, 2016, 24(6): 24–26.
9 刘梦青, 王少辉. 基于蚁群算法的 Storm 集群资源感知任务调度. 计算机技术与发展, 2017, 27(9): 92–96, 100.
10 刘万军, 王晓宇, 曲海成, 等. 基于改进蚁群算法的服务器集群资源调度研究. 微电子学与计算机, 2016, 33(3): 98–101.

© 中国科学院软件研究所 http://www.c-s-a.org.cn