# 一种 Hadoop Yarn 的资源调度方法研究

李媛祯[1],杨 群[1],赖尚琦[2],李博涵[1]
(1. 南京航空航天大学计算机科学与技术学院,江苏南京 210016;2. 香港大学计算机科学与技术系,香港)

**摘 要:** 针对 Hadoop Yarn 资源调度问题,为提高集群作业执行效率,提出一种基于蚁群算法与粒子群算法的自适应 Hadoop 资源调度算法 SRSAPH. SRSAPH 中,通过 Hadoop Yarn 跳通信机制获取负载、内存、CPU 速度等属性信息初始化信息素矩阵;同时,将粒子群算法的自我认知能力与社会认知能力引入到蚁群算法,提高算法的收敛速度;此外,根据蚁群算法全局最优解的波动趋势动态调整信息素挥发系数,提高解的精度. 实验表明,采用 SRSAPH 进行资源调度,集群的作业执行时间缩短至少 10%.

**关键词:** 资源调度; 蚁群算法; 粒子群算法; Hadoop Yarn
**中图分类号:** TP301.6 **文献标识码:** A **文章编号:** 0372-2112 (2016)05-1017-08
**电子学报 URL:** http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.05.002

# A Study on Scheduling Method of Hadoop Yarn

LI Yuan-zhen[1], YANG Qun[1], LAI Shang-qi[2], LI Bo-han[1]
(1. *Computer Science and Technology College, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu* 210016, *China*;
2. *Department of Computer Science, The University of Hong Kong, Hong Kong, China*)

**Abstract:** In view of the resource scheduling problem of Hadoop Yarn, to improve the execution efficiency of the cluster job, we propose a Self-adapt Resource Scheduling algorithm based on Ant Colony Algorithm and Particle Swarm Algorithm in Hadoop (SRSAPH). In SRSAPH, we initialize the pheromone matrix of SRSAPH by using the attribute information of load, memory, and CPU speed obtained through the heartbeat message transfer mechanism. Meanwhile, we introduce the self-cognitive ability and social cognition ability of particle swarm algorithm into the ant colony algorithm to speed up the rate of convergence of the algorithm. Moreover, we dynamically adjust the pheromone evaporation rate based on the fluctuation trends of global optimal solution to enhance the accuracy of the solutions. Experimental result shows that by using SRSAPH in resource scheduling, the execution time of cluster job is shorten by 10%.

**Key words:** resource scheduling; ant colony algorithm; particle swarm algorithm; Hadoop Yarn

## 1 引言

自 2006 年 Yahoo 发布 Hadoop 平台以来,越来越多的应用程序采用 Hadoop 作为处理平台. 要发挥 Hadoop 的作用,合理进行资源分配和调度是关键问题之一[1,2]. 在最新 Hadoop Yarn 中,为解决主节点故障、资源分配不合理等问题,采用资源调度器取代原有的任务调度器,Hadoop Yarn 调度器以层级队列方式组织资源,并让这些队列共享所有节点上的资源. 当节点上存在空闲资源时,节点 NM 向调度器发送空闲资源通知. 调度器实际是一个事件处理器,处理来自其他组件的事件,从而统一管理和调度集群资源,其中,提供了三个内置调度算法,实现了三种资源调度器,即 FIFO, Capacity 和 Fair 调度器. 但是,由于应用的多样性,这些调度器并不能很好地满足用户合理分配资源与减低作业执行时间的需求[3,4].

资源调度问题是一个 NP 难问题,采用传统的调度算法并不能很好地分配系统资源,由于智能算法是求解最优化问题的一种有效手段,为得到资源分配问题的最优解,本文采用智能优化算法设计和实现 Hadoop Yarn 资源调度器以达到降低集群作业执行时间的目标.

## 2 相关工作

由于资源调度的重要性,自 Hadoop 出现以来,工业界和学术界围绕其调度问题进行了大量工作[5~14]. 例如: Sandholm[7]将资源分配过程分解成许多个可以在 NA (Autonomous Node Agents)上以 MCDA (Multiple Criteria

收稿日期:2014-11-15;修回日期:2015-04-09;责任编辑:马兰英
基金项目:国家自然科学基金(No. 41301407);江苏省自然科学基金(No. BK20130819)

Decision Analysis)方式并行执行的独立任务,提高了资源分配的效率,但算法只考虑了硬件资源约束且实现复杂,限制了方法的可扩展性. Zhao[8]提出了一种适用于分布式系统的线性矩阵不等式资源调度算法,但该算法的适用范围小且实际应用意义不大. Ergu[9]提出了一种面向任务的资源调度算法,然而异常元素的不确定性降低了该算法的准确度,尤其是异构环境中,使用偏差矩阵方法易使正常元素误识为异常元素.

相较于上述传统算法,智能优化算法因其求解组合优化问题的有效性,在资源调度领域应用更广. Merkle[12]采用一种基于蚁群算法的资源调度模型,提高了启发式因子在后代信息素挥发变化率上的影响力度,但信息素的大量堆积易造成算法陷入局部最优解而降低算法的寻优能力. Yin[13]的调度策略有效降低算法的停滞性,然而,算法的性能会随着问题复杂度的提高而降低,且该算法仅适应于受限资源环境. Chang[14]提出了一种基于蚁群算法的均衡资源调度算法,通过全局信息素更新与局部信息素更新机制达到资源负载平衡的效果,以便减小作业执行总时间. 但算法将作业执行时间作为已知量参与计算,实际环境下作业执行时间是无法通过现有方法进行精确计算的,这在一定程度上降低了资源分配方案的可信度与准确性.

基于上述分析,本文针对目前最新的 Hadoop Yarn 提出一种基于蚁群算法和粒子群算法[15]的自适应资源调度算法(SRSAPH). 首先,针对蚁群算法虽具有较好的寻优能力但收敛速度慢且易陷入局部最优解的不足,将蚁群算法与粒子群算法认知能力相结合,弥补了蚁群算法在收敛速度慢上的缺陷;然后,根据全局信息素的波动趋势动态调整信息素挥发系数,避免信息素大量堆积,以此提高蚁群算法的全局搜索能力,避免陷入局部最优解,最后设计 SRSARH 算法资源调度器,通过信息素评价集群资源,为每个空闲资源寻求最优的分配策略,从而达到降低集群作业执行时间的目标.

## 3 基于蚁群算法和粒子群算法的自适应 Hadoop 资源调度算法

### 3.1 集群环境描述

集群环境模型表示为 $G=\{A,R,J,C\}$,其中,$A$ 为蚂蚁的集合;$R$ 为节点资源集合;$J$ 为作业集,$C$ 为容器集. 具体特征描述如下:

资源集合 $R=\{R_1,R_2,\cdots,R_n\}$ 由 $n$ 个节点及其资源构成,其中,第 $i$ 个节点上的资源 $R_i=\{cpuSpeed_i,M_i,resM_i,load_i\}$. $cpuSpeed_i$ 为节点 $i$ 上的 CPU 运行速率,节点 $i$ 的总资源量用 $M_i$ 表示,$resM_i$ 为节点 $i$ 的空闲资源量,$resM_i\leqslant M_i$,$0\leqslant i<n$,$load_i$ 表示节点 $i$ 的负载.

作业集 $J=\{J_1,J_2,\cdots,J_s\}$ 表示当前集群上运行的作业量为 $s$,第 $j$ 个作业 $J_j=\{resTotalJ_j,resTotalMJ_j,prog_j\}$,$0\leqslant j<s$,其中,$resTotalJ_j$ 为 $J_j$ 所需资源的总量;$resMJ_j$ 为分配给 $J_j$ 的实际资源量;$prog_j$ 表示 $J_j$ 的运行进度.

容器集 $C=\{C_1,C_2,\cdots,C_t\}$ 由 $t$ 个容器构成,表示集群中所有申请了资源的容器集合,第 $m$ 个容器 $C_m=\{J_{m,j},resMC_m\}$,$0\leqslant j<s$,$0\leqslant m<t$,$J_{m,j}$ 为 $C_m$ 所属的作业,$resMC_m$ 为容器 $C_m$ 所申请的资源量.

### 3.2 SRSAPH 算法模型

本文将蚁群算法应用于资源调度问题,算法各参数与资源调度各参数的对应关系如下.

**定义 1(资源分配)** $\forall R_i\in R,C_m\in C$,如果资源 $R_i$ 可以满足容器 $C_m$ 的调度要求,则记为 $\Theta(R_i,C_m)>0$,表示允许 $R_i$ 为容器 $C_m$ 分配资源以执行任务. 满足容器 $C_m$ 的资源 $R_i$ 可能有多个,为实现集群最大程度的性能优化,$C_m$ 以概率 $p_{i,m}$ 被调度到资源 $R_i$ 上.

**定义 2(信息素矩阵)** 信息素矩阵由一个 $n\times t$ 维矩阵构成,记为 $PH_{n\times t}=(ph_{i,m})_{n\times t}$,其中,矩阵元素 $ph_{i,m}$ $(0\leqslant i<n,0\leqslant m<t$,且 $ph_{i,m}\geqslant 0)$ 表示容器 $C_m$ 在资源 $R_i$ 上所展现出的执行能力,$ph_{i,m}$ 值越大则容器 $C_m$ 在资源 $R_i$ 上的性能越好,反之亦然.

**定义 3(概率矩阵)** 概率矩阵由一个 $n\times t$ 维矩阵构成,记为 $\boldsymbol{P}_{n\times t}=(p_{i,m})_{n\times t}$,,其中,矩阵元素 $p_{i,m}$($0\leqslant i<n,0\leqslant m<t$,且 $p_{i,m}\geqslant 0$)表示容器 $C_m$ 选择在资源 $R_i$ 上启动作业的概率,其值越大则容器 $C_m$ 越倾向于选择资源 $R_i$. 因为同一个容器最多只能选择一个资源,故 $\sum_{m=0}^{t}p_{i,m}=1$.

**定义 4(禁忌矩阵)** 容器禁忌矩阵由一个 $n\times t$ 维矩阵构成,记为 $\boldsymbol{TB}_{n\times t}=(x_{i,m})_{n\times t}$,$0\leqslant i<n,0\leqslant m<t$. $\forall C_m\in C,\exists R_i\in R$,如果 $R_i$ 已为容器 $C_m$ 分配资源,记为 $\Gamma(R_i,C_m)>0$,则矩阵元素 $x_{i,m}=1$,否则 $x_{i,m}=0$.

**定义 5(容器允许矩阵)** 容器允许矩阵由一个 $n\times t$ 维矩阵构成,记为 $\boldsymbol{AT}_{n\times t}=(y_{i,m})_{n\times t}$. 对于矩阵元素 $y_{i,m}$,如果 $\Theta(R_i,C_m)>0$,则 $y_{i,m}=1$,否则 $y_{i,m}=0$. 因为同一个容器最多只能在一个资源上启动,所以容器对应的禁忌矩阵元素与允许矩阵元素之和不能大于 1,该约束条件表示为式(1)、式(2),如下所示. 同时,假设容器 $C_m$ 申请在资源 $R_i$ 上启动,那么容器 $C_m$ 所需资源量不能大于资源 $R_i$ 上的实际空闲资源量,该约束条件表示为式(3),如下所示.

总之,为保证容器的资源请求被顺利响应,需要满足约束条件式(1),(2);为了保证资源 $R_i$ 所接受的容器在其可执行范围内,需要满足约束条件式(3);

$$\sum_{i=0}^{n}x_{i,m}=1 \tag{1}$$

$$\forall x_{i,m}\in \boldsymbol{TB},y_{i,m}\in \boldsymbol{AT},x_{i,m}+y_{i,m}\leqslant 1 \tag{2}$$

$$\forall R_i \in R, \sum_{m=0}^{t} x_{i,m} \cdot resMC_m \leqslant M_i \tag{3}$$

**定义 6(终止条件)** 为保证集群正常运行,文中规定:当算法满足约束条件(4)或(5)时,单个蚂蚁的一次迭代过程结束.

当为 $R_i$ 选择容器分配资源时,为避免 $R_i$ 因无法选择到满足调度条件的容器而陷入死循环,算法增加了控制参数 *attemptContainer*,每次选择容器后 *attemptContainer* 加 1,当算法中控制参数 *attemptContainer* ≥3 或满足约束条件(5)或(6)时,资源 $R_i$ 的分配操作结束.

$$\sum_{i=0}^{n}\sum_{m=0}^{t} y_{i,m} = 0 \tag{4}$$

$$\sum_{i=0}^{n}\sum_{m=0}^{t} x_{i,m} = t \tag{5}$$

$$\sum_{i=0}^{n} y_{i,m} = 0 \tag{6}$$

### 3.3 SRSAPH 算法思想

本文提出的 SRSAPH 算法中,蚂蚁表示调度者,负责分配资源给申请资源的容器以获取资源分配方案.下面详细介绍本文算法的具体流程.

(1)初始化信息素

根据 Hadoop 资源调度框架,本文设计并实现了一个资源调度器,通过该资源调度器从 NM(Node Manager)获取节点 CPU 速率、作业失败记录、内存容量及负载情况等信息,在获取全部相关信息后计算各容器在节点上的执行能力,并将结果以信息素值的形式存储于信息素矩阵,完成信息素矩阵 $\boldsymbol{PH}_{n\times t}=(ph_{i,m})_{n\times t}$ 的初始化. 其中,$ph_{i,m}$ 表示容器 $C_m$ 在资源 $R_i$ 上所展现出的执行能力,由节点 CPU 速率、内存、容器所属作业失败记录的相对权重和乘 $R_i$ 资源剩余利用率计算而来,其表达式如公式(7):

$$phi,m = \left(a * \frac{cpuSpeedi}{\sum_{i=1}^{n}\frac{cpuSpeedi}{n}} + b * \frac{M_i}{\sum_{i=1}^{n}\frac{M_i}{n}} + c * \frac{\psi}{failNote_{m,j}+1}\right) * (1 - load_i) \tag{7}$$

$$load_i = \frac{M_i - resM_i}{M_i} \tag{8}$$

上述公式中:$failNote_{m,j}$ 代表 $C_m$ 所属作业 $J_j$ 在节点 $R_i$ 上的失败记录;$a,b,c$ 均为常数,分别对应节点 CPU 速率、内存、容器所属作业失败记录的权重值,且 $a+b+c=1$. 对于单个节点,作业失败的概率很小,所以 $c$ 的值设定为 0.1. $a,b$ 的取值由作业类型决定:对于计算密集型任务,CPU 对任务执行能力的影响更大,本文设定 $a=0.6, b=0.3, c=0.1$;而对于数据密集型任务,内存对任务执行能力的影响更显著,故设定 $a=0.3, b=0.6, c=0.1$. 对于具体的作业任务可以进一步调节参数以优化调度效果. 节点上任务执行失败因子 $\psi$ 为常数,表示节点上执行失败的任务对后续节点资源的影响.

(2)状态转移概率

一次迭代过程中,资源的分配由多次“节点-容器”操作完成. “节点-容器”操作即为:首先蚂蚁 $ant_k(0\leqslant k<q)$ 随机选择节点资源 $R_i(0\leqslant i<n)$,然后 $ant_k$ 以公式(9)更新状态转移概率矩阵 $P_{n\times t}$,最后以概率 $p_{i,m}(0\leqslant m<t)$ 选择容器,在不满足终止条件时可进行多次选择. 例如,假设当前节点资源 $R_i$ 选中的容器为 $C_m$,若 $\Theta(R_i, C_m)>0$,则 $R_i$ 为 $C_m$ 分配资源同时修改 $x_{i,m}=1$;反之则重新选择容器,直到满足约束条件(5)或(6). $ant_k$ 在 $t$ 时刻选择 $C_m$ 的转移概率为:

$$p_{i,m}(t+1) = \begin{cases} \dfrac{ph_{i,m}^{\alpha}(t)\left[c_1r_1 \mid EP_l - prog_{m,j}(t)\mid\right]^{\beta}\left[c_2r_2 \mid EP_g - prog_{m,j}(t)\mid\right]^{\gamma}}{\sum_{y_{i,h}=1} ph_{i,h}^{\alpha}(t)\left[c_1r_1 \mid EP_l - prog_{h,j}(t)\mid\right]^{\beta}\left[c_2r_2 \mid EP_g - prog_{h,j}(t)\mid\right]^{\gamma}}, & \text{if } y_{i,h}=1, 0\leqslant h<t \\ 0, & \text{else} \end{cases} \tag{9}$$

上式中,针对蚁群算法寻解收敛速度慢问题,改进的状态转移概率机制引入粒子速度更新时的三要素:惯性、自身认知、社会认知. 与传统蚁群算法相比,蚁群算法状态转移概率机制中第一部分信息素量 $ph_{i,m}(t)$ 代表资源 $i$ 相较于容器 $m$ 的信息素量,即“惯性”;而第二部分启发式期望信息素分为两步进行:自身认知期望信息素与社会认知期望信息素. 下面对改进的状态转移概率更新机制中三部分进行详细介绍:

(Ⅰ)$ph_{i,m}^{\alpha}(t)$:蚂蚁资源分配的“惯性”部分,反映了在选择容器时 $R_i$ 受当前信息素的影响能力;$ph_{i,m}(t)$ 代表 $R_i$ 选择 $C_m$ 的信息素;$\alpha$ 为信息启发式因子,表示信息素的相对重要性.

(Ⅱ)$[c_1r_1 \mid EP_l\text{-}prog_{m,j}(t)\mid]^{\beta}$:蚂蚁“自身认知”部分,表示蚂蚁本身的思考;$c_1$ 为加速常数,表示蚂蚁对自身经验的认知能力,调节蚂蚁选择优于自身的容器;$r_1$ 为随机数;$EP_l$ 为蚂蚁寻解所得方案的局部最优解;$prog_{m,j}(t)$ 表示 $C_m$ 所属作业 $J_j$ 的运行进度;$\beta$ 为自我认知期望启发式因子,表示 $prog_{m,j}$ 与局部最优解偏差的自我认知能力,$prog_{m,j}(t)$ 距 $EP_l$ 的偏差越大,$prog_{m,j}(t+1)$ 越大,越倾向于选择当前 $C_m$.

(Ⅲ)$[c_2r_2 \mid EP_g\text{-}prog_{m,j}(t)\mid]^{\gamma}$:蚂蚁“社会认知”部分,表示蚂蚁间社会信息共享;$c_2$ 为加速常数,表示蚂蚁对社会经验的认知能力,调节蚂蚁选择倾向全局最优解发展的容器;$r_2$ 为随机数;$EP_g$ 为蚂蚁寻解所得全局最优解;$\gamma$ 为社会

认知期望启发式因子,表示 $prog_{m,j}(t)$ 与全局最优解偏差的社会认知能力,$prog_{m,j}(t)$ 与 $EP_g$ 的偏差越大,$prog_{m,j}(t+1)$ 越大,越倾向于选择当前 $C_m$.

由公式(9)可知,$ant_k$ 每次选择容器时,若容器 $C_m$ 与资源 $R_i$ 所对应的信息素 $ph_{i,m}$ 越大,则容器被选择的概率越大. 同时,当作业进度 $prog_{m,j}(t)$ 与全局最优解 $EP_g$、局部最优解 $EP_l$ 间的偏差较大时,亦会增加 $C_m$ 被选择的可能性. 这在一定程度上避免了作业进度较低且信息素不具优势的容器长时间不被选择而降低集群作业集进度的情况.

(3)自适应信息素更新机制

当满足终止条件式(4)或式(5)时,蚂蚁 $ant_k$ 一次迭代结束,根据下式(10)计算当前所得解的目标函数 $EP_k$ 的值:

$$EP_k = \sum_{j=0}^{s} [\omega_j * prog_j] \tag{10}$$

$$\omega_j = \frac{\overline{\frac{resMJ_j}{resTotalJ_j}}}{\sum_{j=0}^{s} \overline{\frac{resMJ_j}{resTotalJ_j}}} \tag{11}$$

其中:$\omega_j$ 代表 $J_j$ 运行进度权重值,计算方法如公式(11).

将当前解分别与局部最优解、全局最优解作比较,如果当前解优于局部最优解和全局最优解,则用当前解更新局部最优解和全局最优解,否则不更新. 具体更新机制如下:(1)当迭代次数小于 8 时,挥发系数 $\rho$ 设为定值 0.3[16];(2)当迭代次数大于或等于 8 时,采用自适应信息素更新机制. 如果连续 8 次迭代中全局最优解的变化趋于平稳,适当增大信息素挥发系数,反之减小信息素挥发系数. 以上更新只对本次迭代所得解决方案中包含的资源分配项进行.

上述自适应信息素更新机制可分为下述三步进行:(1)收集 $ant_k$ 最近连续 8 次迭代解决方案作业进度 $EP_k$;(2)计算目标函数的标准差 $\sigma$ 以求得动态挥发系数 $\rho_{ant}$;(3)将 $\rho_{ant}$ 代入公式(14)更新信息素. 接下来详细介绍自适应更新机制.

在获取连续 8 次作业进度 $EP_k$ 后,算法计算目标函数的标准差 $\sigma$,计算公式如下:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{q=1}^{n} [E(p_k)_q - \overline{E(p_k)}]^2}. \tag{12}$$

将 $\sigma$ 代入公式(13)计算动态挥发系数 $\rho_{ant}$,计算公式如下:

$$\rho_{ant} = \frac{\arctan(\varepsilon\sigma)}{\frac{\pi}{2}}, \tag{13}$$

其中,调控参数 $\varepsilon$ 为常量. 若算法计算所得目标函数的标准差较小时,即算法寻解过程趋于稳定,由式(13)计算所得的动态信息素挥发系数 $\rho_{ant}$ 较大($0<\rho_{ant}<1$),反之亦然. 自适应信息素更新机制正是通过调节信息素的挥发量来增强蚂蚁的全局搜索能力,避免蚂蚁陷入局部最优解.

接下来,算法利用 $\rho_{ant}$ 更新信息素,计算公式(14)如下所示:

$$ph_{i,m}(t+1) = \begin{cases} (1-\rho_{ant})ph_{i,m}(t), flag_m = F \\ (1-\rho_{ant})ph_{i,m}(t) + \Delta ph(t), flag_m = T \end{cases} \quad \text{if } \Gamma(R_i, C_m) > 0 \tag{14}$$

$$\Delta ph(t) = \frac{resMC_m}{\sum_{i=0}^{n} resM_i} \tag{15}$$

其中:$\rho_{ant}$ 表示信息素挥发系数;$flag_m$ 表示 $C_m$ 的状态. $\Delta ph(t)$ 表示信息素增量,代表蚂蚁在本次迭代搜索中留在资源分配项上的信息量.

由公式(14)可知,信息素的更新只在 $\Gamma(R_i, C_m) > 0$ 的条件下进行. 这是因为,Hadoop 中,若一个容器所申请资源已获批准,但容器尚未被应用程序管理者注销,此时容器仍会进入下一次资源调度申请,而所申请资源即为零. 当容器所申请资源量为零时,容器状态设为 $F$,反之即为 $T$. 对于 $\Gamma(R_i, C_m) > 0$,若 $flag_m = T$,信息素的更新不仅包括信息素的挥发,还包括信息素积累量的增加,反之只进行信息素的挥发.

由(15)可以看出,算法在分配资源时更趋向于选择大容器. 在分配集群资源时,如果将空闲资源大量分配给小容器会产生大量小容量资源碎片,大容量容器无法搜寻到满足条件的资源量,造成资源浪费. 公式(14),(15)所示的信息素更新方式可以避免上述问题,从而在集群性能允许的范围内最大程度的将资源分配给容器.

## 4 实验

资源调度是一个 NP 难问题,解决此类问题,蚁群算法是一种较优的解决方案[17,18]. 本文在蚁群算法的基础上引入粒子群算法,设计并实现了 SRSAPH 算法. 以下介绍本文实验,并对结果进行分析.

### 4.1 实验环境及参数设置

实验硬件环境为 8 台主机构成的 Hadoop 异构集群,主要参数为:蚂蚁个数 NUM = 20,信息素影响力因子 $\alpha = 1.0$,启发式信息素影响因子 $\beta = 0.8$,$\gamma = 0.8$,加速常数 $c_1 = c_2 = 0.5$,随机数 $r_1 = r_2 = 2$,每次进行资源调度时蚁群算法执行的迭代次数 itrStep = 1000,以上参数均根据前人研究成果,取通用经验值[19]. 节点上任务执行失败因子 $\psi$ 表示同一作业中任务在节点上执行失败的情况,受任务数量与尝试启动次数的限制,故 $\psi$ 取 10. 为使动态挥发系数 $\rho_{ant}$ 能够充分发挥其信息素调整作用,故调控参数 $\varepsilon$ 取 100.

本文选取 4 种 Hadoop 基准实例为测试对象,包括 PI、WordCount、Sort 及 Grep 实例. 文中对比实验的选取依据如下:(1)围绕 Hadoop 资源调度所进行的工作集中在 Hadoop Yarn 发布之前,Hadoop Yarn 发布以后其自身提供的默认调度器更具有代表性;(2)本文所提的算法是基于蚁群算法进行的. 因此,本文对比实验对象为 Hadoop Yarn 提供的默认 Capacity 调度器和前人实现的 Hadoop ACO 调度器(以下简称 ACO 调度器)[20].

本文实验将从下面三方面分析本文 SRSAPH 算法的实验结果:(1)作业集的平均时间、最优时间、最差时间的对比;(2)SRSAPH 算法在作业集执行过程中所占成本分析;(3)作业重复运行 20 次中执行时间的波动趋势. 其中,平均时间取作业集重复运行 20 次的执行时间的平均值.

## 4.2　实验与结果分析

下面对实验结果所获得的数据进行对比与分析.

(1)作业集执行时间对比

**表 1　四种测试对象在不同作业量下的实验结果**

(*a*)PI 实例在不同作业集下的执行时间

| 作业集($N/M_1 * M_2$) | 算法 | 平均时间(s) | 最优时间(s) | 最差时间(s) |
|---|---|---|---|---|
| 4/20 * 20 | SRSAPH | 63.5 | 49.5 | 82.6 |
| | ACO | 64.2 | 48.8 | 89.4 |
| | Capacity | 68.1 | 45.0 | 96.8 |
| 4/100 * 100 | SRSAPH | 203.4 | 193.9 | 220.5 |
| | ACO | 244.5 | 216.1 | 269.2 |
| | Capacity | 262.4 | 224.5 | 298.6 |
| 8/100 * 100 | SRSAPH | 316.4 | 275.9 | 334.4 |
| | ACO | 365.0 | 311.5 | 408.5 |
| | Capacity | 404.5 | 386.6 | 432.3 |
| 8/200 * 200 | SRSAPH | 436.8 | 407.6 | 482.3 |
| | ACO | 485.1 | 428.2 | 501.5 |
| | Capacity | 515.6 | 461.1 | 577.0 |

(*b*)WordCount 实例在不同作业集下的执行时间

| 作业集(个) | 算法 | 平均时间(s) | 最优时间(s) | 最差时间(s) |
|---|---|---|---|---|
| 4 | SRSAPH | 584.5 | 542.4 | 625.6 |
| | ACO | 624.1 | 585.2 | 668.5 |
| | Capacity | 647.2 | 601.9 | 699.2 |
| 8 | SRSAPH | 1042.3 | 969.0 | 1108.7 |
| | ACO | 1092.5 | 994.4 | 1229.1 |
| | Capacity | 1121.4 | 1009.2 | 1443.5 |

(*c*)Sort 实例在不同作业集下的执行时间

| 作业集 | 算法 | 平均时间(s) | 最优时间(s) | 最差时间(s) |
|---|---|---|---|---|
| 512M | SRSAPH | 161.5 | 126.2 | 197.1 |
| | ACO | 169.6 | 144.6 | 190.6 |
| | Capacity | 175.8 | 134.5 | 211.2 |
| 1024M | SRSAPH | 277.5 | 246.1 | 318.4 |
| | ACO | 305.1 | 289.9 | 322.8 |
| | Capacity | 325.5 | 300.5 | 351.2 |
| 2048M | SRSAPH | 525.8 | 485.5 | 544.6 |
| | ACO | 691.1 | 652.8 | 711.8 |
| | Capacity | 754.5 | 684.0 | 859.4 |
| 3096M | SRSAPH | 1381.2 | 1285.5 | 1478.5 |
| | ACO | 1473.8 | 1384.1 | 1599.4 |
| | Capacity | 1575.4 | 1373.7 | 1809.1 |

(*d*)Grep 实例在不同作业集下的执行时间

| 作业集 | 算法 | 平均时间(s) | 最优时间(s) | 最差时间(s) |
|---|---|---|---|---|
| 7 | SRSAPH | 244.7 | 201.4 | 266.3 |
| | ACO | 269.1 | 234.5 | 298.4 |
| | Capacity | 321.8 | 286.5 | 394.2 |
| 14 | SRSAPH | 485.6 | 405.9 | 539.4 |
| | ACO | 535.1 | 524.2 | 566.3 |
| | Capacity | 594.4 | 561.8 | 622.7 |
| 21 | SRSAPH | 812.4 | 776.2 | 904.1 |
| | ACO | 906.1 | 847.4 | 995.2 |
| | Capacity | 1012.9 | 990.7 | 1126.5 |
| 28 | SRSAPH | 1162.5 | 1127.3 | 1233.4 |
| | ACO | 1296.8 | 1240.6 | 1356.4 |
| | Capacity | 1386.4 | 1301.1 | 1516.9 |

表 1 列出 8 节点集群下,采用 SRSAPH、ACO 与 Capacity 调度器处理不同规模的测试对象所展示出的实验性能. 表 1(*a*)中,$N/M_1 * M_2$表示作业集由 $N$ 个 $M_1 * M_2$的作业组成,其中 $M_1 * M_2$表示每单位范围内投掷的点总数,$M_1$为 map 数量,$M_2$为每个 map 中投掷的点数. 表 1(*b*)中,WordCount 实验测试数据为 2013 年 11 月中国新闻汇总,下载自数据堂网站[21]. 每个作业处理一天的数据信息,每一天的信息数据由至少 500 个 txt 文档组成,每个文档对应一个 map 任务,故每个作业在执行时需创建并处理至少 500 个 map. 表 1(*c*)中,利用 RandomWrite 生成不同数量的数据,然后使用 Sort 实例来进行排序. 表 1(*d*)中,Grep 实例测试数据为 7、14、21 与 28 天的新闻数据经,处理后放入 HDFS 指定文档中,然

后对指定文档中指定单词的词频进行计算.

为了更形象的展示不同调度器在集群上的执行效果,现选取表 1(*a*),(*b*),(*c*),(*d*)中平均时间作柱状图,如图 1 中(*a*),(*b*),(*c*),(*d*). 由表 1 可知,与 ACO、Capacity 调度器相比,SRSAPH 的作业集执行时间更短. 在图 1(*a*)中 4/20 * 20 与图 1(*c*)中 512M 情况下,SRSAPH 与 ACO、Capacity 的平均执行时间相差不大,这是因为小作业中的任务执行时间短,且在此情况下集群内存未被完全消耗. 随着作业量的增加,SRSAPH 的优势逐渐体现出来. 如图 1(*a*)中 8/200 * 200、图 1(*b*)中 4 个 WordCount、图 1(*c*)中 3096M 数据集及图 1(*d*)中由 28 天新闻数据组成的数据集,此时集群的负载处于超额状态,此时,集群资源竞争激烈,合理的分配策略可以有效的降低作业的执行时间. 故 SRSAPH 的平均执行时间比 ACO、Capacity 的更短,从而可证明 SRSAPH 算法具有较好的求解效果.

综合表 1 与图 1 可知,在集群负载满额的情况下,Capacity 作业执行时间急剧增长,且 SRSAPH 比 ACO 具有更好的实验性能效果.

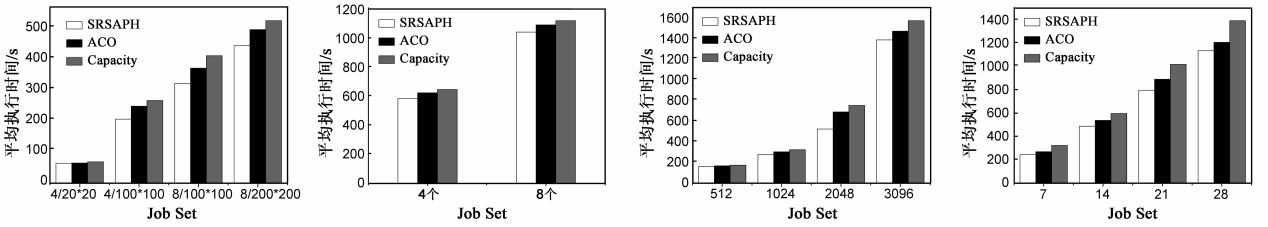
(2)SRSAPH 资源调度成本分析

本文 SRSAPH 调度器下,作业集的执行时间可以分为前期迭代设计资源分配方案时间和后期作业运行时间. 图 2 中作业执行时间与前期额外成本时间均为平均时间. 从图 2(*a*),图 2(*c*)中可以看出,4/20 * 20 下,前期额外成本时间至少为 12s,而 512M 数据下,前期额外成本时间至少为 30s,占总执行时间的比例至少为 20%. 从图 2 可知,(*a*)中 8/200 * 200 作业下额外成本时间所占比例为 31s,(*b*)中 8 个 WordCount 作业下为 62s,(*c*)中 3096M 数据集下为 70s,(*d*)中由 28 天新闻数据组成的作业集下为 65s,占总执行时间的比例最大为 8%. 随着作业的增大,占总执行时间的比例明显减小,且集群作业量在一定范围内时,前期额外成本时间变化不大. 综合图 2 可知,SRSAPH 在处理较大规模作业时所表现出的性能更优.

(3)调度器的稳定性对比

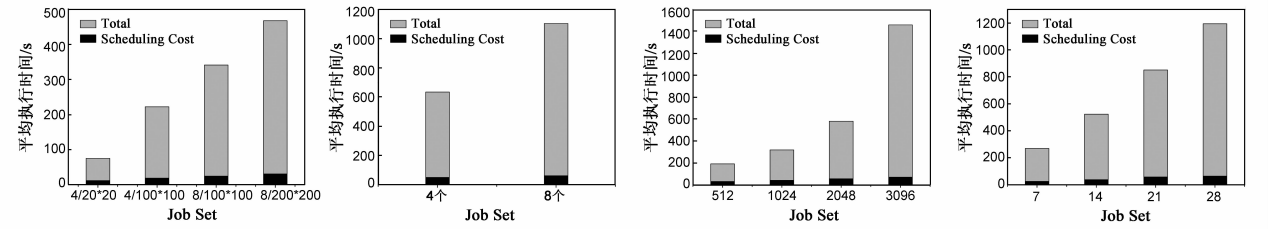
为了观察作业完成时间的波动趋势,了解 SRSAPH 的稳定性,实验分别从 4 种测试对象中选取一组含有任务数量相似的作业,即 8 个 200 * 200 作业、4 个 WordCount 作业、3096M 数据及 28 天新闻数据作业集,在不同调度器条件下运行并分析执行 20 次的时间变化趋势.

由图 3(*a*),(*b*),(*c*),(*d*)可知,随着任务数量的增加,Capacity 的波动幅度增大,图 3(*a*)中 6 点、18 点,图 3(*b*)中 3 点、6 点、7 点、16 点,图 3(*c*)中 6 点、13 点、16 点及图 3(*d*)中 7 点、19 点均为奇异点,在这些点上,Capacity 的执行时间相对较长且与前后点的差值急剧增大,这是因为 Capacity 的每个队列在分配资源时采用的是先到先得方式,若将任务分配给性能较差的节点,则集群作业执行时间较长,反之较短,这时系统作业执行时间所表现出的性能差距较大;而 SRSAPH 与 ACO 采用智能优化算法分配资源,根据集群的实际物理情况,动态调优,使系统资源性能尽量达到最优,但是 SRSAPH 有效的解决了 ACO 中容易陷入局部最优解的缺陷,故 SRSAPH 的波动率始终较小. 故相较于 ACO 与 Capacity,SRSAPH 的作业执行时间基本趋于平稳,最优时间与最差时间波动较小,具有较好的鲁棒性.




(*a*) PI作业平均执行时间对比图 (*b*) WordCount作业平均执行时间对比图 (*c*) Sort作业平均执行时间对比图 (*d*)Grep作业平均执行时间对比图

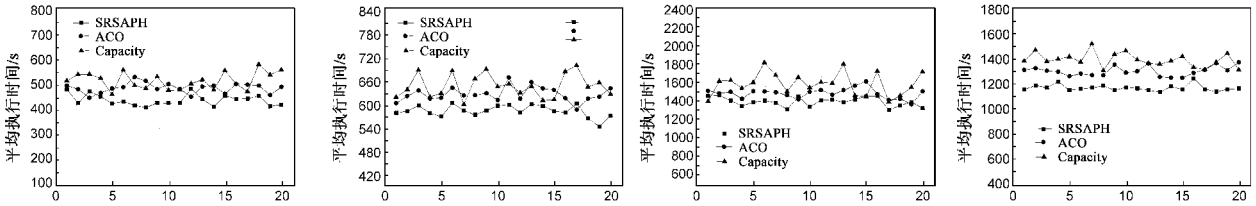
**图1 4种测试对象在不同作业量下平均执行时间对比图**




(*a*) 8个200*200作业下调度成本分析 (*b*) 4个WordCount作业下调度成本分析 (*c*) 3096M作业下调度成本分析 (*d*) 28天新闻数据作业下调度成本分析

**图2 SRSAPH资源调度成本分析图**

从上述实验分析中可以得出,本文 SRSAPH 算法的性能均优于 Capacity 调度器与 ACO 调度器,是一种行之有效的 Hadoop 资源分配策略.




(*a*) 8个200*200作业执行20次变化趋势　(*b*)4个WordCount作业执行20次变化趋势　(*c*)3096M作业执行20次变化趋势　(*d*) 28天新闻数据作业执行20次变化趋势

**图3　SRSAPH调度器稳定性分析图**

## 5　结论

本文针对 Hadoop 环境下资源调度问题,提出了一种基于蚁群算法和粒子群算法的自适应 Hadoop 资源调度算法. 本文算法主要特点表现在:(1)算法较全面的考虑了各个因素对信息素的影响,包括节点内存、负载、CPU 速率及容器所属作业在节点上失败率,对整个集群性能做出较为系统的评估,以此指导蚂蚁资源探索过程;(2)引入粒子群算法更新规则中的三要素:粒子惯性、自身经验、社会经验,吸收了其精度高、收敛快的特点,提高了蚁群算法的收敛速度,结合 Hadoop 环境下资源调度的特性,设计适合 Hadoop 下资源请求与分配的调度器;(3)针对蚁群算法易陷入局部最优解的弊端,本文引入自适应挥发因子,当算法波动较大时,挥发系数较小,而当算法分配方案的波动趋于稳定时,挥发系数增大,从而动态调整挥发系数以避免算法陷入局部最优解. 通过实验证明,根据本文上述思想提出的算法 SRSAPH 资源调度器不仅降低了集群作业执行时间,同时具有较好的鲁棒性.

### 参考文献


[1] Elghoneimy E, Bouhali O, Alnuweiri H. Resource allocation and scheduling in cloud computing[A]. 2012 International Conference on Computing, Networking and Communications[C]. Hawaii: IEEE, 2012. 309 – 314.

[2] Gokilavani M, SelviS, Udhayakumar C. A survey on resource allocation and task scheduling algorithms in cloud environment[J]. International Journal of Engineering and Innovative Technology, 2013, 3(4): 173 – 179.

[3] Chauhan J. Simulation and performance evaluation of hadoop capacity scheduler[D]. Saskatoon: University of Saskatchewan, 2013: 93 – 96.

[4] Fair Scheduler Guide[R/OL]. http://hadoop. apache. org/common/docs/r0. 20. 2/fair _ shcehduler. html, 2013 – 08 – 04.

[5] Rasooli A, Down D G. A hybrid scheduling approach for scalable heterogeneous hadoop systems[A]. 2012 High Performance Computing, Networking, Storage and Analysis[C]. Salt Lake City: IEEE, 2012. 1284 – 1291.

[6] Rasooli A, Down D G. COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems[J]. Future Generation Computer Systems, 2014, 36(7): 1 – 15.

[7] Yazir Y O, Matthews C, Farahbod R, et al. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis[A]. 3th IEEE 2010 International Conference on Cloud Computing[C]. Florida: IEEE, 2010. 91 – 98.

[8] Yue Z, Xu Q. Resource allocation and scheduling theory based on distributed environment[A]. 16th International Conference on Advanced Communication Technology[C]. PyeongChang Korea: IEEE, 2014. 1124 – 1128.

[9] Ergu D, Kou G, Peng Y, et al. The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment[J]. The Journal of Supercomputing, 2013, 64(3): 835 – 848.

[10] Senouci A B, Eldin N N. Use of genetic algorithms in resource scheduling of construction projects[J]. Journal of Construction Engineering and Management, 2004, 130(6): 869 – 877.

[11] Gonçalves J F, Mendes J J M, Resende M G C. A genetic algorithm for the resource constrained multi-project scheduling problem[J]. European Journal of Operational Research, 2008, 189(3): 1171 – 1190.

[12] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(4): 333 – 346.

[13] Yin P Y, Wang J Y. Ant colony optimization for the nonlinear resource allocation problem[J]. Applied mathematics and computation, 2006, 174(2): 1438 – 1453.

[14] Chang R S, Chang J S, Lin P S. An ant algorithm for balanced job scheduling in grids[J]. Future Generation Computer Systems, 2009, 25(1): 20 – 27.

[15] Gandomi A H, Yun G J, Yang X S, et al. Chaos-enhanced

accelerated particle swarm optimization[J]. Communications in Nonlinear Science and Numerical Simulation, 2013,18(2):327 - 340.

[16] Chaharsooghi S K, Meimand Kermani A H. An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP)[J]. Applied Mathematics and Computation,2008,200(1):167 - 177.

[17] Achary R, Vityanathan V, Raj P, et al. Dynamic Job Scheduling Using Ant Colony Optimization for Mobile Cloud Computing[M]. Switzerland: Springer International Publishing,2015:71 - 82.

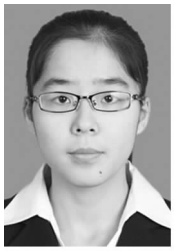
[18] Zhang Z,Zhang N,Feng Z. Multi-satellite control resource scheduling based on ant colony optimization[J]. Expert Systems with Applications,2014,41(6):2816 - 2823.

[19] Dorigo M, Blum C. Ant colony optimization theory: A survey[J]. Theoretical Computer Science,2005,344(1): 243 - 278.

[20] Hengliang S,Guanyi B,Zhenmin T. Aco algorithm-based parallel job scheduling investigation on hadoop[J]. International Journal of Digital Content Technology and Its Applications,2011,5(7):283 - 288.

[21] 数据堂. 2013 年 11 月中国新闻汇总[DB/OL]. http://www. datatang. com/data/45718,2013 - 12 - 23.

## 作者简介



**李媛祯** 女,1990 年出生,硕士研究生,研究方向为资源调度、并行计算.
E-mail:liyuanzhen0724@163. com

**杨 群(通信作者)** 女,1971 年出生,副教授,博士,研究方向为新型程序设计、软件方法学.
E-mail:qun. y@163. com