

基于蚁群优化算法的 MapReduce 集群调度策略

秦 军¹, 张建平², 王 昊², 魏家宾²

(1. 南京邮电大学 教育科学与技术学院, 江苏 南京 210003;

2. 南京邮电大学 计算机学院, 江苏 南京 210003)

摘 要:在云计算环境中, MapReduce 集群已成为强大的大规模数据集处理平台。针对其在任务调度过程中存在用户 QoS、集群资源利用率等方面的缺陷, 提出了一种基于蚁群优化算法的调度策略(ACO-SS)。该调度策略同时考虑了优先级计算模型和任务调度过程, 能有效地满足用户 QoS, 平衡集群节点负载, 使分布在节点上的任务利用资源更加合理, 提高了系统的调度性能。最后, 通过 CloudSim 仿真实验表明, 该调度策略在作业完成总体时间、资源利用率等重要指标上都具有明显优势。

关键词: MapReduce 集群; 蚁群优化算法; 用户 QoS; 调度策略

中图分类号: TP31

文献标识码: A

文章编号: 1673-629X(2013)06-0074-05

doi: 10.3969/j.issn.1673-629X.2013.06.019

Scheduling Strategy for MapReduce Cluster Based on Ant Colony Optimization Algorithm

QIN Jun¹, ZHANG Jian-ping², WANG Hao², WEI Jia-bin²

(1. College of Education Science & Technology, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;

2. College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: MapReduce cluster has become a powerful processing platform for large-scale data sets in cloud computing. For the defects in task scheduling process, such as users' QoS and the utilization of cluster resources, a scheduling strategy named ACO-SS which is based on ant colony optimization algorithm is proposed. With ACO-SS scheduling strategy, the priority calculation model and scheduling process have been taken into account. So users' QoS can be effectively satisfied, and the load balance of nodes are strengthened, and it's more rational that the resources are used by tasks distributed on nodes, and the system's scheduling performance is improved. Finally, the simulation experiments with Cloudsim toolkit show that ACO-SS has obvious advantages in these important indexes, such as job's whole completion time and resource utilization etc.

Key words: MapReduce cluster; ant colony optimization algorithm; users' QoS; scheduling strategy

0 引 言

在云环境下, MapReduce(MR)作为分布式计算模型主要是用来处理和产生大规模数据集^[1]。在 MR 模型里主要包含一个主节点 JobTracker 和若干个工作节点 TaskTracker。其中, JobTracker 的工作是通过不断的监听 JobClient 来等待作业的提交, 并将用户提交的作业划分为多个 Map 和 Reduce 任务, 而任务的执行主要由 TaskTracker 来完成。据调研, 目前国内外学者对 MR 集群相关的调度策略研究的也比较多。对于作

业的调度, Zaharia 等人提出了作业延时等待的策略^[2], Matei 等人提出了一种公平性调度算法^[3], Polo 等人提出了根据作业执行的时间来限制作业的调度^[4]等。对于资源的调度, Buyya 等人提出了利用调度策略^[5]来评估作业性能、分析资源成本, Polo 等人还提出了资源自适应的调度机制^[6]。以上文献中提出的调度算法在一定的范围内具有优势, 但是针对混合作业、集群资源利用率等方面仍然存在一定的局限性。针对以上问题, 文中在考虑到包括数据本地性问题、用户

收稿日期: 2012-09-03

修回日期: 2012-12-08

网络出版时间: 2013-05-14

基金项目: 江苏省自然基金项目(BK2009425); 江苏省教育科学十二五规划课题(D/2011/01/074)

作者简介: 秦 军(1955-), 女, 江苏南京人, 教授, 主要从事计算机网络技术、数据库技术研究; 张建平(1987-), 男, 重庆巫山人, 硕士研究生, 主要研究方向为云计算、分布式计算。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20130514.1707.003.html>

QoS 等多方面后重点探讨了 MR 集群中的调度机制。在此基础上,提出了一种基于蚁群优化算法的调度策略 (Ant Colony Optimization algorithm - Scheduling Strategy, ACO-SS)。

1 形式化描述

定义1 作业集 $J = \{J_1, J_2, \dots, J_i, \dots, J_n\}$, 其中 J_i 代表一个作业。每个作业划分为对应的一组 Map 任务集 $T = J_i = \{t_1, t_2, \dots, t_k, \dots, t_m\}$, 其中 t_k 代表一个任务。

定义2 设 $V = \{v_1, v_2, \dots, v_n\}$ 表示拥有不同处理能力的工作节点构成的节点集, 而每个工作节点 v_i 拥有处理速度 p 、内存容量 r 、CPU 数目 m 和网络传输带宽 b 四个度量参数, 它们可以综合反应出 v_i 的原有处理能力。分别为这四个度量参数设置阈值 P_0 、 R_0 、 M_0 、 B_0 , 若超过阈值, 则统一以阈值计算。构建工作节点的初始信息素:

$$\begin{aligned} \tau_{v_i}(0) = & \left\{ \alpha \left(\frac{m * p}{M_0 * P_0} \times 100\% \right) + \beta \left(\frac{r}{R_0} \times 100\% \right) + \chi \left(\frac{b}{B_0} \times 100\% \right) \right. \\ & \left. \alpha + \beta + \chi = 1 \right. \end{aligned} \quad (1)$$

定义3 在任务调度时, MR 调度器会计算出任务分配到请求节点(请求节点即请求分配任务的工作节点)上的初始概率, 即在 t 时刻, 任务 t_k 分配到请求节点 v_i 上的初始概率由公式(2)确定。

$$p^{t_k}(t, v_i) = \begin{cases} \frac{\tau^{\alpha}(t, v_i) * \eta^{\beta}(t, v_i)}{\sum_{j \in U} \tau^{\alpha}(t, v_j) * \eta^{\beta}(t, v_j)}, & i, j \in \text{有效的节点集 } U \\ 0 & \text{其他} \end{cases} \quad (2)$$

式中, $\tau(t, v_i)$ 表示在时刻 t , 任务 t_k 在工作节点 v_i 上的信息素浓度。 $\eta(t, v_i)$ 为工作节点 v_i 的原始能力即 $\eta(t, v_i) = \tau(0, v_i) = \tau_{v_i}(0)$ 。 α 和 β 分别是衡量 τ 和 η 相对重要性的参数。

定义4 设任务 t_k 在工作节点 v_i 上的预计执行时间耗费为 $\text{Cost}(t_k, v_i)$; 任务 t_k 分配到 v_i 的网络传输时间为 $\text{Delay}(t_k, v_i)$ 。 $\text{Time}(t_k, v_i)$ 表示任务 t_k 在 v_i 上的完成时间, 等于执行时间与网络传输时间之和, 即 $\text{Time}(t_k, v_i) = \text{Cost}(t_k, v_i) + \text{Delay}(t_k, v_i)$ 。 $\text{TaskListLength}(v_i)$ 表示工作节点 v_i 中待执行的任务队列长度, 其大小为队列中所有任务完成的时间之和。结合定义3 中的初始分配概率可构成最终的分配概率模型如公式(3), 即选择处理能力强和任务队列较短的请求节点进行任务分配。

$$\text{MaxP} =$$

$$\begin{cases} \lambda_1 p^{t_k}(t, v_i) + \frac{\lambda_2}{\text{TaskListLength}(v_i)} \\ \text{s. t.} \begin{cases} \text{TaskListLength}(v_i) = \sum_{k=1}^n \text{Time}(t_k, v_i) \\ \lambda_1, \lambda_2 \text{ 为权重比值} \end{cases} \end{cases} \quad (3)$$

2 基于蚁群优化算法的 ACO-SS 调度策略

蚁群优化算法是模拟蚂蚁觅食过程而设计出的一种种群智能算法, 主要用于求解组合优化问题, 其典型范例是经典旅行商 TSP 问题^[7]。ACO-SS 的调度过程主要是通过研究蚁群优化算法为任务进行节点调度, 并通过新建优先级模型, 增加任务池和激励因子来优化调度。

2.1 优先级模型建立

Hadoop 是对 MR 模型的开源实现, 它是开源软件基金会组织 Apache 提供的一个开源云计算平台^[8]。Hadoop 中可控的调度优先级只到 Job 级别, 考虑到数据本地性问题, ACO-SS 调度在此基础上新增 Task 优先级使可控调度达到了 Task 级别。在 MR 模型中, 系统处理任务的能力可以用系统同时处理 Map 任务数的能力来表示, 记为 TaskCapacity。而每个 Map 任务的数据量为 HDFS^[9] 的分块大小, 记为 Blocksize。假设用 W 表示一次调度中所有作业的数据总量, 那么 W 可表示为:

$$W = \text{Max} \left(\frac{\text{TaskCapacity} * \text{Blocksize}}{\text{JobNum} * \text{AvgTasknum} * \text{Blocksize}} \right) \quad (4)$$

式(4)中, JobNum 表示一次调度中执行的作业总数, AvgTasknum 表示每个作业平均执行的任务数。若用 AvgTaskSize $[i]$ 来表示任务的平均大小, 那么其值可以用作业 i 的大小 JobSize $[i]$ 与执行作业 i 需要的任务数 TaskNum $[i]$ 相除得到。同时, 若用 Priority 表示作业设定的初始优先级值, Weight $[i]$ 表示根据 Priority 获得的可执行的任务数, 那么在一次调度中需处理作业 i 的数据量即 $S[i]$, 等于任务的平均大小 AvgTaskSize $[i]$ 与 Weight $[i]$ 的乘积。因此, 作业 i 的优先级与所有作业优先级之和的比值应该等于一次调度中作业 i 所处理的数据量与所有作业的数据总量的比值, 即有:

$$\frac{\text{Priority}[i]}{\sum_{i=1}^n \text{Priority}[i]} = \frac{S[i]}{W} \quad (5)$$

由(5)式可推导出 Weight $[i]$ 的值, 在某种程度上说, Job 优先级即 Jobpriority 的大小也就是 Weight 的值, 可表示为:

$$\text{Jobpriority} = \text{Weight}[i] =$$

$$\frac{W * Priority[i]}{AvgTaskSize[i] * \sum_{i=1}^n Priority[i]} \quad (6)$$

同样, Task 优先级也主要针对 Map 任务, 主节点 JobTracker 需要为每一个 Map 任务计算调度优先级, 其大小设置为 Taskpriority。Taskpriority 的计算公式如(7)所示:

$$Taskpriority = \frac{\alpha}{TaskNodeNum} + \beta * (TaskListLength) \quad (7)$$

式中, 设置调节因子 α 为 0.6, β 为 0.8。TaskNodeNum 表示可以本地执行 Map 任务的工作节点数目, TaskListLength 表示执行 Map 任务的工作节点的本地任务队列长度。以上重点分析了 Jobpriority 和 Taskpriority 的求解过程, 作业最终的优先级计算模型如公式(8):

$$JTprioritv = \begin{cases} \text{MaxValue} \\ \delta * Jobpriority + \eta * Taskpriority \end{cases} \quad (8)$$

式中, δ, η 为调节因子。其中, 若 JTprioritv 的值设置为 MaxValue, 则此作业被优先执行, 否则按照上述公式进行计算。

2.2 任务调度过程设计

针对用户的 QoS 描述通常可以采用整体完成时间、网络带宽等参数指标来量化^[10], 文中选取执行单个作业的总时间作为评判指标。ACO-SS 中新增了一个带有 FIFO 性质的任务池 Taskpool, 用来记录正处理的作业和对应分解的 Map 任务, 它由 MR 调度器来维护, 在获得作业之后调度器会将该作业和分解的任务一起添加到 Taskpool 中, 而后根据算法进行调度。

设定信息素分布在工作节点上并且将主要的计算和传输量作为信息素的求解对象。随着任务的执行, 工作节点上的信息素也会发生相应的变化, 并根据任务的执行情况来更新信息素。在 t_1 时刻将任务 t_k 分配给请求节点 v_i 时, 信息素按照公式(9)进行更新:

$$\tau_{v_i}(t_1) = \tau_{v_i}(t) - \rho \tau_{v_i}(t), \rho \in [0, 1] \quad (9)$$

在任务执行了一段时间后, 无论执行是否成功, 系统的负载都会得到一定程度的减轻。因此为了平衡节点的负载, 信息素的浓度会相应的增长。同时, ACO-SS 中还引入了激励因子 ε , 若在当前工作节点上成功运行完任务则进行激励, 否则给予惩罚, 见公式(10)。

$$\tau_{v_i}(t_2) = \begin{cases} (1 + \varepsilon)(\tau_{v_i}(t_1) + \rho_1 \tau_{v_i}(t_1)), \rho_1 \in [0, 1] \\ 0 < \varepsilon < 1, \text{ 成功} \\ -1 < \varepsilon < 0, \text{ 失败} \end{cases} \quad (10)$$

若作业 J_k 分解的所有 Map 任务都被处理完毕后, 将更新一次所有处理过该作业 Map 任务的工作节点

的信息素。同时, 为了加快收敛速度, 对于未处理过该作业 Map 任务的工作节点的信息素予以削弱。其中 c_e, ρ_2 为调节因子常量, $Time(J_k)$ 表示作业 J_k 的完成时间, 对于工作节点 v_i 和作业 J_k 有:

$$\tau_{v_i}^k(t_2) = \begin{cases} \tau_{v_i}^k(t_1) + \Delta \tau_j^k \\ \Delta \tau_j^k = \begin{cases} \frac{c_e}{Time(J_k)}, \text{ 节点 } v_i \text{ 处理过 } J_k \text{ 的 Map 任务} \\ -\rho_2 * \tau_{v_i}^k(t_1), \text{ 其他} \end{cases} \end{cases} \quad (11)$$

在整个调度过程中, 如果工作节点在一段时间内未得到任何任务的分配, 那么该节点的信息素将按照公式(12)进行挥发, 其中 $1 - \rho_3$ 表示挥发系数。

$$\tau_{v_i}^k(t_2) = (1 - \rho_3) \tau_{v_i}^k(t_1), \rho_3 \in [0, 1] \quad (12)$$

2.3 算法实现过程

针对用户提交的作业定义了 JobActiveQueue[] 队列, 其主要作用是维护按优先级排序的作业队列。ACO-SS 的基本调度流程描述如下:

步骤 1 Jobclient 向 JobTracker 提交一批作业, 系统根据 2.1 节提出的优先级计算模型为每个作业计算优先级权值, 并进行排序后添加到 JobActiveQueue[] 队列中。

步骤 2 系统根据公式(1)初始化各个 TaskTracker 工作节点信息素, 当检测到集群中有 TaskTracker 向 JobTracker 发出请求任务分配的消息, MR 调度器将所有请求节点的 ID 号加入到禁忌表 Tabu list 中。同时从 JobActiveQueue[] 中取出队首作业, 并将其与对应分解的 Map 任务一起添加到 Taskpool 中。

步骤 3 根据公式(2)计算出 Tabu list 里面每一个请求节点的初始转移概率, 并从 Taskpool 里面取出任务, 根据任务情况计算出公式(3)的最终转移概率 P , 选择 Max P 进行转移, 并且将被调度的请求节点 ID 号从禁忌表 Tabu list 中删除。

步骤 4 当任务正常分配给 TaskTracker 后, 利用公式(9)和(10)进行相应的信息素更新。如果任务执行失败, 则重新插入到属于本作业在 Task pool 中的位置, 等待下一次重新调度。

步骤 5 从 Taskpool 中取出下一个任务进行工作节点调度, 重复步骤 3、4。

步骤 6 若当前一个作业处理完毕, 则根据公式(11)更新处理过该作业 Map 任务的工作节点的信息素, 同时削弱那些未处理过该作业 Map 任务的工作节点的信息素。

步骤 7 根据公式(12)挥发信息素。

步骤 8 从 JobActiveQueue[] 中取出下一个作业

添加到 Taskpool 中,重复步骤 2~7。

3 仿真分析及性能评价

CloudSim^[11]是澳大利亚墨尔本大学网络实验室和 Gridbus 项目推出的云计算仿真软件。文中将使用 CloudSim 对调度策略 ACO-SS 进行模拟仿真,在模拟仿真实验中将 ACO-SS 与公平调度算法(FSA)^[12]和资源自适应调度算法(RASA)在评价指标方面进行比较。

3.1 模拟环境

创建实验所需的虚拟集群网络,集群由 12 台虚拟机和 3 个用户构成,详细的性能配置参数见表 1。设置 ID=1 为主控节点 JobTracker,另外 11 台为工作节点 TaskTracker。

表 1 集群性能配置参数

ID	PE	CPU 数目	MIPS	Brand(Mb/S)	Memory
1	1	2	600	40	2G
2	1	2	300	40	1G
3	1	2	280	40	512M
4	2	4	380	40	1G
5	2	4	390	40	2G
6	3	2	350	40	1G
7	3	4	400	40	1G
8	4	2	360	40	512M
9	4	2	355	40	1G
10	4	2	375	40	512M
11	5	2	275	40	2G
12	5	4	385	40	2G

3.2 参数设置

为初始信息素的求解设置 $M_0=8, P_0=1024\text{MIPS}$, $R_0=8\text{G}, B_0=40\text{Mb/s}$ 。ACO-SS 中需要设置相关公式的参数,详细的设置情况见表 2。

表 2 仿真参数设置

	α	β	χ	λ_1	λ_2	ρ	ρ_1	ρ_2	ρ_3	ε	c_e
节点 CPU 的影响因子	0.4										
内存的影响因子	0.3										
网络带宽的影响因子	0.3										
权重比值	0.4 0.6										
信息素 τ 的相对重要性	1										
η 的相对重要性	1										
信息素更新调节因子	0.3 0.5 0.1 0.7 + / - 0.2 1										

3.3 仿真结果

通过运行多次大小不同的作业求解 JTpriority 中

调节因子 δ, η 的最优组合。选择大小分别为 2G, 4G, 8G 的作业,重复运行 20 次,最后得出有 65% 的作业在 $\delta=1, \eta=2$ 时,其完成时间最短,是最优组合。实验生成 1~8G 的 8 个作业,其数据大小如表 3 所示。

根据表 1 和表 3 提供的数据,可以计算出任务在节点上的预计执行时间 Cost,对于 Delay 的时间设定在 $[100, 400]\text{ms}$ 之间随机产生。实验对单个用户进行测试,用户提交 100 个作业,而每个作业的大小在 $[\text{Job1}, \text{Job8}]$ 内随机产生。共进行 20 次实验分别执行 ACO-SS、FSA 和 RASA 三种算法,取其平均作业完成时间值,结果如图 1 所示。

表 3 作业测试数据

	Job1	Job2	Job3	Job4	Job5	Job6	Job7	Job8
JobSize	1G	2G	3G	4G	5G	6G	7G	8G
Blocks	16	32	48	64	80	96	112	128
Priority	1	1	1	1	1	1	1	1
MapTasks	16	16	32	32	64	96	96	64
ReduceTasks	1	1	1	1	1	1	1	1

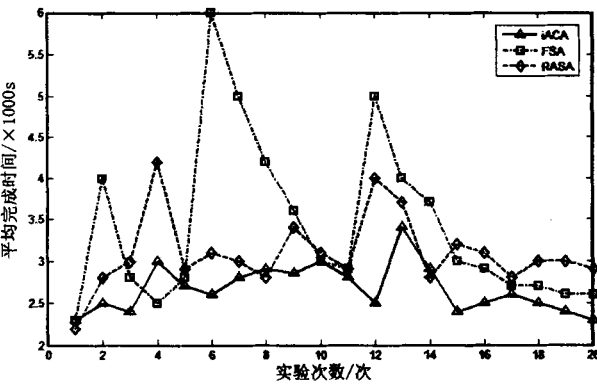


图 1 不同实验次数下的作业平均完成时间

从图 1 中可以发现,在不同的实验次数下,ACO-SS 的作业平均完成时间整体上比 FSA 和 RASA 要少。如在实验次数为 20 时,ACO-SS 的平均完成时间为 2400s 左右,比 FSA 算法减少约 200s,比 RASA 算法减少约 400s。随着实验次数的不断增多,三种算法的作业平均完成时间也趋于稳定,而 ACO-SS 的优势也相对较为明显,平均减少了约 9% 左右的完成时间,很好地满足了用户 QoS。再次提交 200 个作业,分别让集群执行 10~60 分钟,三个算法在资源利用率和节点负载方面的比较如图 2 和图 3 所示。

从图 2 中可以发现,在不同执行时间段的情况下,相对于 FSA 和 RASA,ACO-SS 的资源利用率都相对较高。特别是当执行时间越长即执行的作业数量越多时,其资源利用率提升的更加明显,如在执行 60 分钟后,ACO-SS 比 FSA 提升了 10% 左右,比 RASA 提升了 8% 左右。在图 3 中,FSA 和 RASA 两种算法在节点 3 和 11 的负载比较轻,而在节点 10 和 12 的负载又比较重,而 ACO-SS 策略能很好地改善这种负载不均衡的

情况。从模拟结果可看出,ACO-SS 在用户 QoS、集群利用率以及平衡节点负载等方面具有更好的优势。

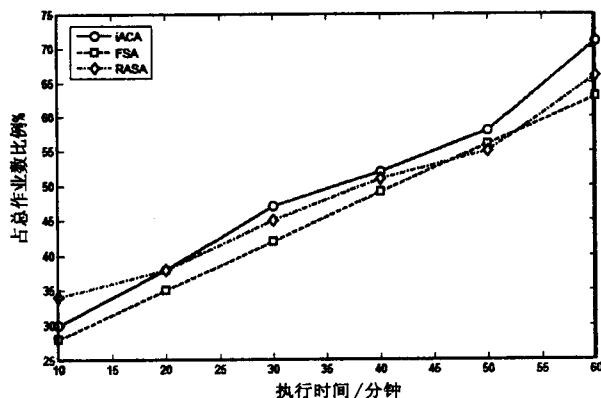


图2 不同执行时间段下的资源利用率

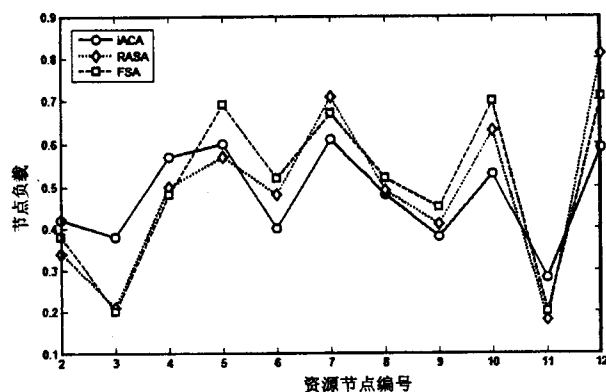


图3 资源节点负载比较

4 结束语

文中针对 MapReduce 集群提出了一种基于蚁群优化算法的调度策略,该策略不但把作业总体完成时间作为参考指标,而且把集群资源利用率和节点负载也直接作为整体性能优化的参考指标。该策略能够很好地满足用户 QoS,节点负载均衡以及集群资源利用率。该调度对于 MapReduce 集群来说,是一种十

分实用有效的调度策略。

参考文献:

- [1] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[C]//Proc of Sixth Symposium on Operating System Design and Implementation. Berkeley: USENIX Association, 2004: 137-150.
- [2] Zaharia M, Borthakur D, Sarma J S. Job scheduling for multi-user Mapreduce clusters[C]//Proceedings of the 5th European Conference. Washington: IEEE Computer Society, 2009: 145-161.
- [3] Zaharia M, Borthakur D, Sarma J S. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]//Proceedings of the 5th European Conference on Computer Systems. New York: ACM, 2010: 265-278.
- [4] 顾宇,周良,丁秋林.基于优先级的 Three-Queue 调度算法研究[J].计算机科学, 2011, 38(10): 253-256.
- [5] 刘永,王新华.云计算环境下基于蚁群优化算法的资源调度策略[J].计算机技术与发展, 2011, 21(9): 19-23.
- [6] Jorddal P, Claris C, David C, et al. Resourceaware adaptive scheduling for MapReduce clusters[C]//Middleware 2011 - ACM/IFIP/USENIX 12th International Middleware Conference. New York: ACM, 2011: 187-205.
- [7] 段海滨.蚁群算法原理及其应用[M].北京:科学出版社, 2006: 15-20.
- [8] Apache Hadoop[EB/OL]. 2012-04-16. <http://hadoop.apache.org/>.
- [9] 郝树魁. Hadoop HDFS 和 MapReduce 架构浅析[J].邮电设计技术, 2012, 21(9): 37-42.
- [10] 李振东,谢立. Web 服务器群的 QoS 确保及其接纳控制研究[J].计算机研究与发展, 2005, 42(4): 662-668.
- [11] CloudSim[EB/OL]. 2012-02-11. <http://www.cloudbus.org/cloudsim/>.
- [12] Hadoop 公平调度算法[EB/OL]. 2010-02-19. http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html.


(上接第 73 页)

- [3] Bennett J, Zhang Hui. WF2Q: Worst-case Fair Weighted Fair Queueing[C]//Proc of IEEE INFOCOM 96. [s. l.]: IEEE Press, 1996: 120-128.
- [4] Demers A, Keshav S, Shenkar S. Analysis and simulation of a fair queueing algorithm[C]//Proc of SIGCOMM '89. New York: ACM, 1990.
- [5] Amrami B. WFQ and WF2Q[J]. Course: Topics in MultiProcessing, 2001, 25(8): 122-153.
- [6] Wang Song. Hierarchical Qos integration for realtime systems[D]. California: University of California, 2003.
- [7] Abhay K, Parekh A. Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case[C]//Proc of IEEE INFOCOMM 92. [s. l.]: IEEE

Computer Society Press, 1992: 915-924.

- [8] 李方敏,李仁发,欧育立.路由队列管理机制[J].计算机工程, 2001, 27(8): 222-232.
- [9] 王重钢,隆克平,龚向阳.分组交换网络中队列调度算法的研究及其展望[J].电子学报, 2001, 29(4): 53-59.
- [10] Shreedhar M, Varghese G. Efficient fair queueing using deficit roundrobin[J]. IEEE/ACM Transactions on Networking, 1996, 4(3): 375-385.
- [11] Bennett J C R, Zhang H. WF2Q: Worst-case fair weighted fair queueing[C]//Proc of IEEE INFOCOMM 96. San Francisco, CA: [s. n.], 1996: 120-128.
- [12] NS2 教学手册[EB/OL]. 1996. <http://www.isi.edu/nsnam/ns/doc/index.html>

基于蚁群优化算法的MapReduce集群调度策略

作者：秦军， 张建平， 王昊， 魏家宾， QIN Jun， ZHANG Jian-ping， WANG Hao， WEI Jia-bin
作者单位：秦军, QIN Jun(南京邮电大学教育科学与技术学院, 江苏南京, 210003)， 张建平, 王昊, 魏家宾, ZHANG Jian-ping, WANG Hao, WEI Jia-bin(南京邮电大学计算机学院, 江苏南京, 210003)
刊名：计算机技术与发展 
英文刊名：Computer Technology and Development
年， 卷(期)：2013, 23(6)

本文链接：http://d.g.wanfangdata.com.cn/Periodical_wjz201306019.aspx