# 基于蚁群算法的 Storm 集群资源感知任务调度

刘梦青[1,2],王少辉[1,2]
(1. 南京邮电大学 计算机学院,江苏 南京 210003;
2. 江苏省无线传感网高技术研究重点实验室,江苏 南京 210003)

**摘 要**:实时计算系统 Storm 是当前十分流行的开源流式系统,在处理流式数据时具有明显的优势,但也存在默认调度器在任务调度时难以将节点资源与任务需求相结合、节点资源利用率不高、节点内存不足以及网络堵塞等问题。为了解决这些问题,提出了一种基于蚁群算法的 Storm 集群资源感知任务调度算法及其实现方案。该算法将节点的资源动态变化表示为蚂蚁运动所需的信息素,将任务调度过程模拟为蚂蚁觅食过程,以此对任务调度进行优化,保证了 Storm 任务调度的有效性。实验结果表明,该算法能够找到与当前任务所需资源最匹配的节点,从而实现资源的合理分配;与默认调度相比,具有更优的任务调度效率、更少的平均处理时间和更高的集群吞吐量,有利于集群负载均衡,优化集群的性能。

**关键词**:Storm;资源感知;蚁群算法;负载均衡
**中图分类号**:TP39 **文献标识码**:A **文章编号**:1673-629X(2017)09-0092-05
doi:10.3969/j.issn.1673-629X.2017.09.020

# Research on Storm Resource-aware Task Scheduling with Ant Colony Algorithm

LIU Meng-qing[1,2], WANG Shao-hui[1,2]
(1. College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;
2. Key Laboratory of High Technology Research for Wireless Sensor Networks of Jiangsu Province, Nanjing 210003, China)

**Abstract**: Storm is the popular open source real-time computing system, which has a great advantage in handling data stream. However, there are some problems in its default scheduler when scheduling tasks, such as difficultly in combining node resources and mission requirements, ineffectiveness in node resource utilization, lack of memory and network congestion and so on. In order to solve them, a resource-aware scheduler based on ant colony algorithm and its implementation scheme is proposed, in which the dynamic changes of the node resource can be expressed as the pheromones of ant movement required and the task scheduling process is similar to ant foraging process, to optimize the task scheduling and ensure the effectiveness of the Storm task scheduling. Experimental results show that it has found the most suitable node for the current task and achieved the reasonable allocation of resources and that compared with the default scheduling, it has better task scheduling efficiency, less average processing time and higher throughput of the cluster, which can benefit the load balance and optimize the performance for the cluster.

**Key words**: Storm; resource-aware; ant colony algorithm; load balance

## 1 概 述

随着互联网的快速发展和云计算等技术的兴起,数据正以前所未有的速度暴增,数据处理问题成为了当前不可忽视的问题。其中以多源并发、数据汇聚、在线处理为特征的流式数据(Data Stream),已经成为当前的研究热点。Storm 是当前十分流行的分布式流式数据处理系统[1],其强大的分布式集群管理、便捷的针对流式数据的编程模型、高容错非功能保障,是它成为业界主流的首要原因。

为了能快速有效地处理数据,Storm 集群中任务调度的合理设置十分关键。任务调度是 Storm 集群的重要组成部分,集群里有独立存在的默认调度器(Scheduler)。集群在用户提交作业(Topology)后,启动调度器,将作业分配到工作节点中执行。Storm 集

**收稿日期**:2016-10-01 **修回日期**:2017-02-14 **网络出版时间**:2017-07-11
**基金项目**:国家自然科学基金资助项目(61572260);江苏省科技支撑计划项目(BE2015702)
**作者简介**:刘梦青(1992-),女,硕士研究生,研究方向为信息系统的安全与隐私;王少辉,博士,副教授,研究方向为信息安全、密码学。

群的默认调度策略在判断节点资源是否足够时,更关注节点的 CPU 资源,而忽略内存、磁盘、网络等其他类型的节点资源,这样有可能造成工作节点发生内存不足、网络堵塞等问题。另外,默认调度器没能和任务的实际需求相结合,导致在任务调度的过程中,未能取得很好的调度效果。

对 Storm 默认调度算法改进的研究受到越来越多研究者的关注。Aniello 等[2]设计了在线自适应调度器,通过监控作业运行时间,减少节点间的通信来改善调度。该方案只在作者设计的作业中有效,不具备普遍性。Long 等[3]针对作业的实际场景的不同来改进调度算法,如恢复历史调度任务、单节点任务调度、资源需求调度等。Wang 等[4]提出多层调度算法,通过减少组件的长尾时延(the long tail delay)来提高任务调度的效率。Eskandari 等[5]提出了自适应分层调度算法,使得资源分配更加有效,并且提高了集群的性能。虽然这些改进对提高 Storm 集群的资源管理和任务调度具有一定的效果,但是并未考虑任务需求和当前集群资源相结合的问题,这是研究的着眼点。

任务调度是典型的 NP-hard 问题,通过模仿自然界生物行为特征的智能优化算法,是解决这类问题的有效方法。常见的智能化任务调度算法有遗传算法、微粒群算法、蚁群算法等[6],其中蚁群算法在资源感知方面具有显著的优势。为此,针对 Storm 集群默认调度不能对任务进行合理调度(即合理分配资源)的问题,结合蚁群算法对默认调度进行优化,提出了基于蚁群算法的 Storm 资源感知任务调度算法。

## 2　Storm 默认调度策略

如图 1 所示,在 Storm 集群中,用户提交的程序称为 Topology,表现为有向无环图,由 Spout 和 Bolt 构成。Spout 和 Bolt 统称为 component,在集群中运行实例的单位是 task。Topology 启动后,component 的 task 数目固定不变。

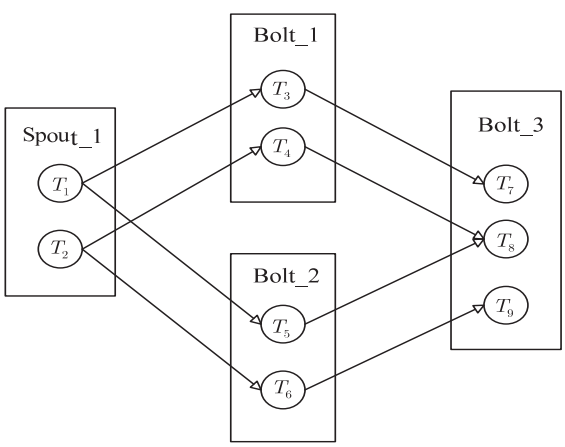


图 1　用户提交的 Topology

Storm 集群由一个主控节点 Nimbus,多个协调节点 Zookeeper 和多个工作节点 Supervisor 组成,每个工作节点上都配置有 slot 数[7]。Storm 的默认调度器 Scheduler 在主控节点上,负责为用户提交上来的 Topology 分配资源,将任务分配到工作节点上。默认调度器的任务分配调度策略如下:

(1)在集群机器 slot 资源足够的情况下,能均匀地将所有 Topology 的 task 分配到整个集群的所有工作节点上;

(2)当 slot 资源不够时,会将所有 Topology 的 task 全部分配到仅有的 slot 上,由有限进程处理这些任务,此时的分配不理想。一旦出现新的空闲 slot,又会重新分配 Topology 的 task,以达到理想的状态;

(3)没有空闲 slot,Nimbus 什么也不做。

以图 1 的 Topology 为例,假设此时集群有三个工作节点 Supervisor,并且节点资源足够,当用户提交 Topology 后,默认调度器对任务的分配情况如下。

(1)Supervisor1:$T_1$, $T_4$, $T_7$;

(2)Supervisor2:$T_2$, $T_5$, $T_8$;

(3)Supervisor3:$T_3$, $T_6$, $T_9$。

用户提交的 Topology 中的 components 可分成三类:输入层、计算层、输出层,而每一层对资源的需求都不一样[8]:

(1)输入层:如果数据来自于磁盘,则需要更多的磁盘资源;若来自于网络,则需要更多的带宽资源。

(2)计算层:该层任务的特点是要进行大量的计算,消耗 CPU 资源,需要更多的 CPU 和内存。

(3)输出层:类似于输入层,可能需要更多的磁盘将数据存入本地或者是更多网络带宽将数据传输给别的服务器。

Storm 默认调度器在调度时没能将集群节点所拥有的资源和任务的实际需求相结合,以致没有取得很好的任务调度效果。另外,采用默认调度,在判断资源分配是否均匀时,几乎只考虑了节点的 CPU 使用情况,而节点其他类型的资源,比如内存、磁盘、网络等未做考虑。

若一个 component 属于 I/O 密集型的输入层,这种类型任务的特点是对网络、磁盘资源消耗很多,CPU 资源消耗很少,这类任务执行期间 99% 的时间都花在 I/O 上,花在 CPU 上的时间很少。按照默认调度器,将其分配到某个工作节点上,很有可能该节点拥有的资源更多是 CPU,而不是该 component 最需要的 I/O 资源,这必然会造成不合理的资源分配现象。也就是说,默认调度器实际上并不能均匀有效地分配资源,可能会造成 Supervisor 节点发生内存不足、网络堵塞等问题,对集群的性能造成严重影响。

## 3 基于蚁群算法的 Storm 任务调度

蚁群算法由 Marco Dorigo 等于 1991 年提出[9],该算法是对自然界蚂蚁的寻径方式进行模拟而得出的一种仿生算法。经过 20 多年的发展,蚁群算法广泛应用于车间调度问题[10]、车辆路径问题[11]、分配问题、决策支持以及仿真和系统辨别等领域,为这些 NP-hard 的组合优化问题的解决提供了有效且高效的办法。利用蚁群算法来解决 Storm 集群默认调度策略不能对资源进行合理分配的问题,提出基于蚁群算法的 Storm 资源感知任务调度算法,将 Storm 任务调度过程效仿蚂蚁觅食过程,将任务比作蚂蚁,对任务调度的过程进行优化。

### 3.1 Storm 任务调度问题描述

在 Storm 平台上,将一个 Topology 中的 $n$ 个 tasks 分配到 $m$ 个 Supervisors 工作节点上执行($m < n$),初始描述为:$n$ 个任务集表示为 $T = \{t_1, t_2, \cdots, t_n\}$,其中 $t_i (i = 1, 2, \cdots, n)$ 表示第 $i$ 个 task;用 $SV = \{sv_1, sv_2, \cdots, sv_m\}$ 表示 $m$ 个 Supervisor 节点集,其中 $sv_j (j = 1, 2, \cdots, m)$ 表示第 $j$ 个 Supervisor 节点;设 $et_{ij}$ 为任务 $t_i$ 在节点 $sv_j$ 上的预测完成时间,则整个 Topology 的 tasks 分配到 Supervisor 节点上的预测完成时间可组成矩阵 $\mathbf{ET}_{[n,m]}$:

$$\mathbf{ET} = \begin{bmatrix} et_{11} & et_{12} & \cdots & et_{1m} \\ et_{21} & et_{22} & \cdots & et_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ et_{n1} & et_{n2} & \cdots & et_{nm} \end{bmatrix} \tag{1}$$

其中,$et_{ij}$ 为第 $i$ 个任务在第 $j$ 个 Supervisor 节点上的预测完成时间,假设 Topology 的每个 task 的完成时间已经预测得到。

### 3.2 算法描述

提出算法基于蚁群算法的思想,通过效仿蚂蚁觅食的过程,将任务比作蚂蚁,对任务调度的过程进行优化,当蚂蚁找到食物时,也意味着完成了任务分配。算法中通过计算节点的信息素浓度来决定分配给某任务的节点,并且总是选择信息素最浓的节点即资源丰富的节点进行分配。该算法的执行步骤如下:

Step1:初始化算法参数,设置各 Supervisor 节点的信息素;

Step2:设置算法的最大迭代次数;

Step3:将 $n$ 只蚂蚁随机发送到 $m$ 个节点上;

Step4:每只蚂蚁根据预测时间阈值限制和节点选择概率选择下一节点;

Step5:当所有任务都分配完后,更新全局信息素,否则跳转至 Step4;

Step6:算法达到最大循环次数,输出最优解,否则跳转至 Step3。

算法流程如图 2 所示。

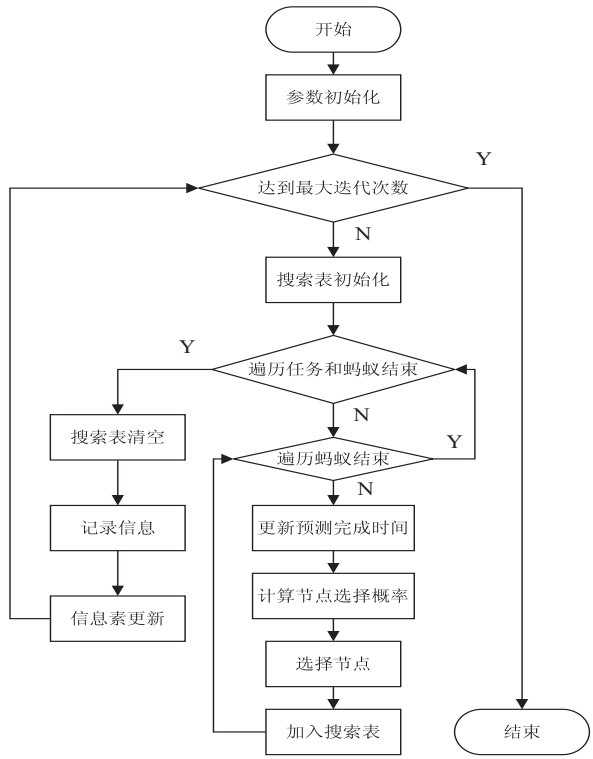


图 2 基于蚁群算法的 Storm 任务调度流程

下面给出算法中涉及的一系列参数的设定方法。

(1)Supervisor 节点的信息素表示。

在 Storm 中,Supervisor、Worker、executor 等组件的心跳信息会同步至 Zookeeper,Nimbus 会周期性地利用 Zookeeper 上关于 Supervisor 的心跳信息得到 Supervisor 节点 $sv_i (i = 1, 2, \cdots, m)$ 的可用资源情况。分别用 $c$, $m$, $d$, $n$ 代表节点的 CPU、内存、磁盘和网络的信息素,并且每个参数的阈值定为 $c_0$, $m_0$, $d_0$, $n_0$,即节点可提供资源的最大值,用于防止节点超负载。可将信息素初始化为:$\tau_{ic}(0) = c/c_0$;$\tau_{im}(0) = m/m_0$;$\tau_{id}(0) = d/d_0$;$\tau_{in}(0) = n/n_0$。

节点 $i$ 的信息素是各信息素的带权和,其中 $a$, $b$, $c$, $d$ ($a + b + c + d = 1$)分别表示不同任务所需 CPU、内存、磁盘、网络资源的权重。例如,对于内存密集型任务,可适当增加内存信息素所对应的权重 $b$,此时,内存资源相对丰富的节点更容易被选中。

$$\tau_i(t) = a\tau_{ic}(0) + b\tau_{im}(0) + c\tau_{id}(0) + d\tau_{in}(0) \tag{2}$$

在任务调度的过程中,很有可能出现这样的情况:某些节点因为具备最优资源(即最浓信息素),一直被分配任务,始终处于过于忙碌状态;而那些信息素浓度较低的节点可能一直都没有分配到任务而处于闲置状态,这样也会造成负载不均衡的现象。为了解决这个问题,增加控制负载系数 $s = S_c/S_{sum}$,$S_c$ 表示已经完成

的任务,$S_{sum}$ 表示任务的总量。因此,节点的信息素更新公式为:

$$\tau_i = s \times \tau_i(t) \tag{3}$$

(2)任务预测完成时间。

在使用蚁群调度时,需计算出所有节点上每个任务的预测完成时间,生成 **ET** 矩阵,并将该矩阵作为蚁群算法的启发信息之一。随着任务不停地分配完成,还没完成的任务随之会减少,任务完成时间也会发生变化,因此,各任务的预测完成时间的更新公式定义为:

$$\begin{aligned} \mathrm{ET}_j^{\,n\,\mathrm{predict}}(J_{\mathrm{predict}}(t_2)) = & \frac{n_{\mathrm{predict}}}{n_{\mathrm{previous}}} \times \\ & (\lambda \mathrm{ET}_j^{\,n\,\mathrm{previous}}) J_{\mathrm{previous}}(t_0) + \\ & (1-\lambda)\mathrm{ET}_j^{\,n\,\mathrm{previous}}(J_{\mathrm{previous}}(t_1)) \end{aligned} \tag{4}$$

其中,$\mathrm{ET}_j^{\,n\,\mathrm{predict}}(J_{\mathrm{predict}}(t_2))$ 表示在 $t_2$ 时刻新任务 $J_{\mathrm{predict}}$ 在 $j$ 节点上的预测完成时间;$n_{\mathrm{predict}}$ 表示在 $t_2$ 时刻 $j$ 节点上运行的任务数。

另外,为了判断节点是否为有效节点,设置了一个有效区间,如果新任务的预测完成时间在该区间内,则该节点为有效节点;如果节点是没有分配过任务的节点,则需要根据节点的资源自动生成任务的预测完成时间,如果该时间在有效区间内,则节点为有效节点。

(3)蚁群算法节点的选择。

在算法运行过程中,蚂蚁会根据启发信息及信息素的浓度进行节点间的转移。蚂蚁 $h$ 由节点 $i$ 转移到节点 $j$ 的状态转移概率公式[12]如下:

$$p_{ij}^h = \begin{cases} \dfrac{[\tau_{ij}]^\alpha [\eta_j]^\beta}{\sum\limits_{j \notin \mathrm{tabu}_h} [\tau_{ij}]^\alpha [\eta_j]^\beta}, j \notin \mathrm{tabu}_h \\ 0, j \in \mathrm{tabu}_h \end{cases} \tag{5}$$

其中,$p_{ij}^h$ 为蚂蚁 $h$ 由节点 $i$ 转移到节点 $j$ 的概率;$\tau_{ij}$ 为在节点 $i$ 观察到节点 $j$ 的信息素浓度;$\eta_j = 1/\mathrm{et}_j$,$\mathrm{et}_j$ 为任务 $t_j$ 的预期执行时间;$\alpha$ 为信息启发式因子,表示 $\tau_{ij}$ 的重要性;$\beta$ 为期望启发式因子,表示 $\eta_j$ 的重要性;$\mathrm{tabu}_h(h = 1,2,\cdots,m)$ 为蚂蚁 $h$ 已经走过的节点,即不允许再走的节点,称做禁忌表,集合 $\mathrm{tabu}_h$ 随着算法运行进行动态调整。

经过一段时间后,所有蚂蚁都遍历了所有的有效节点,并且有属于自己的路径 $L_h$,将蚂蚁当前所在节点列入禁忌表,计算时间最短的路径 $L_{h\min}(\min L_h, h = 1,2,\cdots,m)$,在该路径上选择信息素最浓的节点,将任务分配给该节点。

(4)信息素更新。

当有新的任务分配到节点上时,节点的资源被消耗,信息素值会随之减少,此时信息素更新如下:

$$\tau_i(t+n) = \tau_i(t) - \rho\tau_i(t), 0 < \rho < 1 \tag{6}$$

其中,$\tau_i(t+n)$ 为 $t+n$ 时刻新任务到达 $i$ 节点上的信息素浓度;$\tau_i(t)$ 为节点 $i$ 在时刻 $t$ 的信息素浓度;$\rho$ 为调节因子。

当节点上的任务执行完成(成功或者失败)时,节点的资源便会被释放,信息浓度也随之增加,公式如下:

$$\tau_i(t+m) = \tau_i(t+n) + \rho_1\tau_i(t+n), 0 < \rho_1 < 1 \tag{7}$$

另外,增加一个调节因子 $\rho_2$,用来鼓励成功执行任务的节点或者是惩罚执行任务失败的节点,达到引导蚂蚁行为的目的。

$$\tau_i(t+m) = (1+\rho_2)(\tau_i(t+n) + \rho_1\tau_i(t+n)) \tag{8}$$

如果执行任务成功,则 $0 < \rho_2 < 1$;如果执行任务失败,则 $-1 < \rho_2 < 0$。

## 4　实验及分析

在 Storm0.8.0 版本之后,Storm 提供了可插拔的调度器(Pluggable Scheduler)[13],可用于自定义任务的分配调度算法,以实现特定需求。该实验利用 Pluggable Scheduler 实现自定义的调度。另外,用 Ganglia[14]来监控 Storm 集群的各节点状态,如:CPU、内存、硬盘利用率,I/O 负载,网络流量等。

实验中,集群的环境配置由 5 台物理机器组成,硬件配置如表 1 所示。

表 1　集群硬件配置信息

| 节点名称 | 硬件配置 |
|---|---|
| Master | 4 核 2.7 GHz CPU,4 GB 内存,20 GB 硬盘,10 Gbit 网卡 |
| Slave1 | 4 核 2.7 GHz CPU,4 GB 内存,20 GB 硬盘,10 Gbit 网卡 |
| Slave2 | 4 核 2.7 GHz CPU,4 GB 内存,20 GB 硬盘,10 Gbit 网卡 |
| Slave3 | 4 核 2.7 GHz CPU,4 GB 内存,20 GB 硬盘,10 Gbit 网卡 |
| Slave4 | 4 核 2.7 GHz CPU,4 GB 内存,20 GB 硬盘,10 Gbit 网卡 |

主控节点 Master 上运行 Nimbus 和 Zookeeper 守护进程,从节点 Slave 上运行 Supervisor 守护进程,每个节点上的系统为 Ubuntu 12.0.4,每个节点上配置 4 个 slot。

相对于现有的改进方案,文中算法旨在改善任务需求和当前集群资源相结合中存在的问题,故设计了三个对资源种类需求不一样的 Topology。通过观察执行情况来分析该算法的使用情况。Topology_1 属于内存密集型作业,Topology_2 属于网络密集型作业,To-

pology_3 属于 CPU 密集型作业。数字 1 ~ 10 代表资源使用量,各作业的估计资源使用值如表 2 所示。

表 2 各 Topology 估计资源使用值

| Topology | CPU | Memory | Disk | Network |
|---|---|---|---|---|
| Topology_1 | 5 | 8 | 5 | 4 |
| Topology_2 | 2 | 4 | 5 | 8 |
| Topology_3 | 9 | 5 | 4 | 3 |

先将 3 个不同 Topology 分别提交到 Storm 集群中,它们的平均处理时间如图 3 所示。可以发现,在提交上去的三个作业中,开始阶段节点可提供的资源都一样,改进调度算法不如默认调度算法,执行一段时间后,节点的可提供资源发生变化,这时改进调度算法表现得比默认算法要好很多,相对于默认调度算法,改进调度算法会将作业平均处理时间减少 56.8% 左右,并且随着时间的推移逐渐稳定,对提高作业的执行效率具有很好的效果。接着,将三个 Topology 都提交到集群中,分别调用默认调度算法和改进调度算法对任务进行分配,最后收集每个 Topology 的吞吐量,结果如图 3 所示。可以发现,和默认调度算法相比,改进调度算法可以明显提高 Topology 的吞吐量,集群的整体吞吐量大概提高了 37% 。

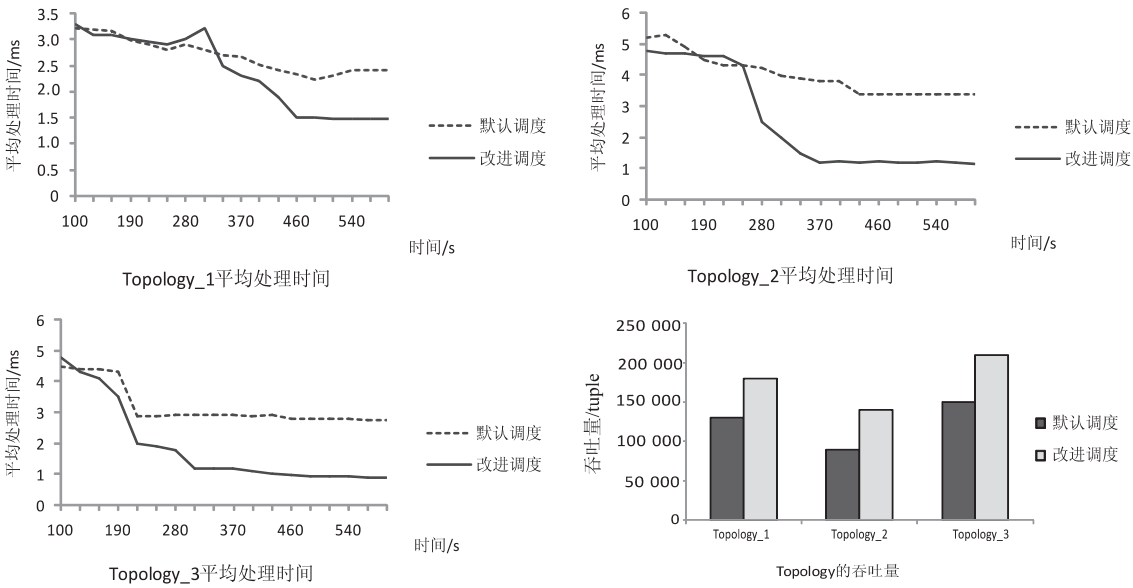


图 3 实验结果

另外,从 Ganglia 的监控数据可以看到,在调用改进调度算法的过程中,集群中节点的 CPU、内存、磁盘等使用都较为均衡,没有出现过忙或过闲的节点,整个集群的负载较为均衡。

由实验结果可知,所提出的基于蚁群算法的资源感知调度算法,在调度执行过程中节点资源和任务的实际需求资源更契合,不仅考虑到节点的 CPU,还考虑到了节点的内存、磁盘、网络等资源,在减少 Topology 的平均处理时间和提高吞吐量方面具有比默认调度算法更好的表现,并且在 Topology 执行期间,集群节点没有超负载,集群负载较为均衡。

## 5 结束语

围绕 Storm 集群任务调度问题,针对 Storm 默认调度不能解决不同任务在资源需求和占用上的差别和不能有效利用节点资源的问题,提出一种基于蚁群算法的资源感知算法。该算法将感知到的集群节点资源作为信息素,将要进行分配执行的任务比作蚂蚁,任务分配的过程类似蚂蚁寻食的过程。资源越丰富的节点分配到更多的任务,优化了任务分配过程,降低了任务平均完成时间,提高了集群的吞吐量。该算法在任务调度初期会耗费一些时间搜集信息素并进行迭代操作,因此如何减少时间是进一步研究的方向。



**参考文献:**

[1] 丁维龙,赵卓峰,韩燕波. Storm:大数据流式计算及应用实践[M]. 北京:电子工业出版社,2015:27-33.

[2] Aniello L, Baldoni R, Querzoni L. Adaptive online scheduling in storm[C]//Proceedings of the 7th ACM international conference on distributed event-based systems. [s. l.]: ACM, 2013:207-218.

[3] Long S, Rao R, Miao W, et al. An improved topology schedule algorithm for storm system[C]//Proceedings of the 2014 Asia-Pacific conference on computer science and applications. [s. l.]: CRC Press, 2014:187-192.

[4] Wang J, Hang S, Liu J, et al. Multi-level scheduling algorithm based on Storm[J]. KSII Transactions on Internet & Informa-

(下转第 100 页)

$\frac{1}{N-1}\sum_{i=1}^{N}(g_i-\bar{g})$。

通过上面的公式计算 MSR 算法和神经网络算法处理之后图像各自对应的值,见表 1。

表 1 客观指标对比结果

| 图像 | MSR 算法 | | | 神经网络增强算法 | | |
|---|---|---|---|---|---|---|
| | PSNR | $K$ | $Q$ | PSNR | $K$ | $Q$ |
| boy | 29.53 | 0.45 | 0.42 | 30.33 | 0.50 | 0.40 |
| painting | 28.46 | 0.43 | 0.39 | 29.95 | 0.46 | 0.41 |
| dog | 30.45 | 0.51 | 0.47 | 41.25 | 0.55 | 0.51 |

表 1 中,神经网络算法的 PSNR 和 $K$ 都大于 MSR 算法的 PSNR 和 $K$ 的值,这两个值越大,表示增强后的图像质量就越好,因此神经网络图像增强算法在图像细节方面更加丰富,图像清晰度更高,视觉方面更逼真。综合这些指标可知,相对于 MSR 算法,神经网络算法对图像有较好的增强效果,也具有一定的优越性。

## 3 结束语

图像增强技术一直是计算机视觉领域中的研究焦点。针对当前的 Retinex 算法在工作中的局限性,为获得更好的图像增强效率,提出了 RBF 神经网络的图像增强算法。实验结果表明,相对于 Retinex 算法,在给定图像评价指标的情况下,RBF 神经网络的高反差图像增强方法具有较好的优势。神经网络算法可以保留丰富的细节信息,获得更清晰的图像,实时性强、算法简单灵活,具有广阔的应用前景。

**参考文献:**


[1] 严 义,吴迎笑. 基于神经网络的图像高反差算法的研究与实现[J]. 仪器仪表学报,2006,27:2302-2305.

[2] Kim Y T. Contrast enhancement using brightness preserving bi-histogram equalization[J]. IEEE Transactions on Consumer Electronics,1997,43(1):1-8.

[3] 李学明. 基于 Retinex 理论的图像增强算法[J]. 计算机应用研究,2005,22(2):235-237.

[4] 王 焱,关南楠,刘海涛. 改进的多尺度 Retinex 井下图像增强算法[J]. 辽宁工程技术大学学报:自然科学版,2016,35(4):440-443.

[5] 张立明. 人工神经网络的模型及其应用[M]. 上海:复旦大学出版社,1993.

[6] 李明国,郁文贤. 神经网络的函数逼近理论[J]. 国防科技大学学报,1998,20(4):70-76.

[7] Españaboquera S,Zamoramartínez F,Castrobleda M J,et al. Efficient BP algorithms for general feedforward neural networks[C]//International work-conference on the interplay between natural and artificial computation. [s. l. ]:Springer-Verlag,2007:327-336.

[8] Yu B,He X. Training radial basis function networks with differential evolution[C]//IEEE international conference on granular computing. [s. l. ]:IEEE,2006:369-372.

[9] 殷 勇,邱 明. 一种基于高斯核的 RBF 神经网络学习算法[J]. 计算机工程与应用,2002,38(21):118-119.

[10] Krishna K,Murty M N. Genetic K-means algorithm[J]. IEEE Transactions on Cybernetics,1999,29(3):433-439.

[11] 刘颖超,张纪元. 梯度下降法[J]. 南京理工大学学报:自然科学版,1993(2):12-16.

[12] 卫 敏,余乐安. 具有最优学习率的 RBF 神经网络及其应用[J]. 管理科学学报,2012,15(4):50-57.

[13] 李 杰,韩正之. 神经网络的学习误差函数及泛化能力[J]. 控制与决策,2000,15(1):95-97.

[14] 佟雨兵,张其善,祁云平. 基于 PSNR 与 SSIM 联合的图像质量评价模型[J]. 中国图象图形学报,2006,11(12):19-24.


(上接第 96 页)


tion Systems,2016,10(3):1091-1110.

[5] Eskandari L,Huang Z,Eyers D. P-Scheduler:adaptive hierarchical scheduling in apache storm[C]//Australasian computer science week multiconference. [s. l. ]:ACM,2016:26-31.

[6] 钟一文. 智能优化方法及其应用研究[D]. 杭州:浙江大学,2005.

[7] Toshniwal A,Taneja S,Shukla A,et al. Storm@ twitter[C]//ACM SIGMOD international conference on management of data. [s. l. ]:ACM,2014:147-156.

[8] Dena D, Bucicoiu M, Bardac M. A managed distributed processing pipeline with Storm and Mesos[C]//International conference on networking in education and research. [s. l. ]:IEEE,2013:1-6.

[9] Dorigo M, Maniezzo V, Colomi A. The ant system:optimization by a colony of cooperation agents[J]. IEEE Transactions on Systems,Man and Cybemetics,1996,26(1):29-41.

[10] 黄亚平,熊 婧. 基于改进蚁群算法作业车间调度问题仿真研究[J]. 计算机仿真,2009,26(8):278-282.

[11] Barán B,Schaerer M. A multi objective ant colony system for vehicle routing problem with time windows[C]//21st IASTED international multi-conference on applied informatics. [s. l. ]:[s. n. ],2003:97-102.

[12] 李德启,田素贞. 一种基于云环境下蚁群优化算法的改进研究[J]. 陕西科技大学学报:自然科学版,2012,30(1):64-68.

[13] Playmud. Twitter Storm 的新利器 Pluggable Scheduler[EB/OL]. 2012-05-21. http://blog. chinaunix. net/uid-233938-id-3216108. html.

[14] Massie M L,Chun B N,Culler D E. The ganglia distributed monitoring system: design, implementation, and experience[J]. Parallel Computing,2004,30(7):817-840.