文章编号:1671-4598(2019)08-0236-05 DOI:10.16526/j.cnki.11—4762/tp.2019.08.049 中图分类号:TP306.2 文献标识码:A

改进蚁群算法的 Storm 任务调度优化

王 林,王 晶
(西安理工大学 自动化与信息工程学院,西安 710048)

摘要:Apache Storm 默认任务调度机制是采用 Round—Robin(轮询)的方法对各个节点平均分配任务,由于默认调度无法获取集群整体的运行状态,导致节点间资源分配不合理;针对该问题,利用蚁群算法在 NP—hard 问题上的优势结合 Storm 本身拓扑特点,提出了改进蚁群算法在 Storm 任务调度中的优化方案;通过大量实验找到了启发因子 α 与 β 的最佳取值,并测得改进后蚁群算法在 Storm 任务调度中的最佳迭代次数;引入 Sigmoid 函数改进了挥发因子 ρ,使其可以随着程序运行自适应调节;从而降低了各个节点 CPU 的负载,同时提高了各节点之间负载均衡,加快了任务调度效率;实验结果表明改进后的蚁群算法和 Storm 默认的轮询调度算法在平均 CPU 负载上降低了 26%,同时 CPU 使用标准差降低了 3.5%,在算法效率上比 Storm 默认的轮询调度算法提高了 21.6%。

关键词:Storm;任务调度;蚁群算法;负载均衡

Task Scheduling Optimization of Storm Based on Improved Ant Colony Algorithm

Wang Lin, Wang Jing
(School of Automation and Information Engineering, Xi' an University of Technology, Xi' an 710048, China)

Abstract: Apache Storm's default task scheduling mechanism uses Round—Robin (Polling) to distribute tasks to each node evenly. The default scheduling cannot obtain the overall running state of the cluster, resulting in unreasonable resource allocation between nodes. Aiming at this problem, the advantages of ant colony algorithm on NP—hard problem combined with the topology characteristics of Storm itself are proposed. The optimization scheme of improved ant colony algorithm in Storm task scheduling is proposed. The optimal values of heuristic factors α and β were found by a large number of experiments, and the optimal number of iterations of the improved ant colony algorithm in Storm task scheduling was measured. The Sigmoid function was introduced to improve the volatilization factor ρ, so that it can be used with the program. Run adaptive adjustment. Thereby reducing the load of each node CPU, and improving load balancing between nodes, speeding up task scheduling efficiency. The experimental results show that the improved ant colony algorithm and Storm's default polling scheduling algorithm reduce the average CPU load by 26%, while the CPU standard deviation is reduced by 3.5%. The algorithm efficiency is higher than Storm's default polling scheduling algorithm 21.6%.

Keywords: Storm; task scheduling; ant colony algorithm; load balance

0 引言

随着大数据时代的到来和互联网的飞速发展,我们生活中的一切都被数据所记录,这些数据量已经超过了我们的想象。为了分析和处理这些数据,学者们提出了许多种类的分布式框架,如 Hadoop 中的 MapReduce 框架等,但是大多数框架都是批处理框架。尽管这些框架擅长批处理,但其处理速度都难以保证延时低于 5 秒。这类批处理框架无法满足许多项目当中对结果需要实时反馈的要求,这时 Storm 流计算框架应运而生。Storm 是一个免费开源、分布式、高容错的实时计算系统,可以使持续不断的流计算变得容易,弥补了 Hadoop 批处理所不能满足的实时要求;Storm 经常用于实时分析、在线机器学习、持续计算、分布式远程调用和 ETL 等领域;Storm 的部署管理简单,而且 Storm 对于流数据处理的优势也是其它框架无法比拟的。

Apache Storm 的默认任务调度机制是采用 Round—Robin(轮询)的方法,即 Storm 提交的拓扑作业按照总的任务数量平均分配给各个节点。其忽略了不同节点间的性能、负载及节点间的通信问题,使得各个节点间的资源不能充分利用。结果导致 Storm 工作时效率下降。所以 Storm 能够合理快速的处理数据,其任务调度非常重要。针对 Storm 任务调度存在的优化问题,国内外已有学者对其进行研究。文献[1]提出一个实现资源系统感知的资源调度优化算法 R—Storm,R—Storm 是通过把资源分为两类,针对内存的计算资源和网络的计算资源,根据两种资源对不同节点的分配来实现调度任务,最终实现资源利用率最大化、提高总体吞吐量同时最小化网络延迟。文献[2]提出一个根据流量感知的资源调度优化算法 T—Storm,T—Storm 利用加速数据处理用于分配和重新分配任务的有效流

收稿日期:2019-02-22; **修回日期**:2019-03-16。
基金项目:陕西省科技计划重点项目(2017ZDCXL—GY—05—03)。
作者简介:王 林(1963-),男,江苏省东台人,硕士生导师,教授,主要从事复杂网络、信息安全、大数据方面的研究。

量感知调度，动态地减少节点间和进程间的流量，同时确保没有工作节点过载。文献［3］提出一种实时高效资源调度和优化框架，称为重新流（Re－Stream），重新流描绘了能量消耗和响应之间的数学关系，利用分布式模型对数据流图进行建模，重新分配利用高效的启发式和关键路径调度机制完成任务调度优化。文献［4］提出了一种低复杂度的随机调度方法和优化框架—云集群中的图，这些图表随着时间的推移而变化，进而任务调度的优化。文献［5］提出一种分层阶段调度算法，该算法是通过降低进程和进程节点与节点之间的开销来完成优化调度。文献［6］提出了一种基于 Topology 的优化调度策略。该策略比 Storm 自带的默认调度策略拥有更高的流事件处理效率，且实施在分支结构较多的 Topology 上效果比 Storm 默认调度策略更优。文献［7］提出了基于启发式均衡图划分算法的调度策略，通过对 Storm 建立调度模型，将负载检测作为调度器的输入实现动态并行参数优化和重调度优化，最终减少集群节点间的数据发送率，并且保持节点间负载均衡。虽然以上这些算法对任务调度都有一定的效果，但是与各自还有些不足存在。

文献［1，2，3］中的算法中的配置完全依赖工作人员去自己设定，不能通过算法自身反馈来获得，不利于于实际应用；文献［4］中的算法不能完全移植到 storm 系统当中；文献［5］没有考虑到自身计算开销的大小；文献［6］中的算法没有考虑跨界点传输时的通信问题；文献［7］再划分图时，没有考虑虑顶点权重，负载均衡上存在问题。在已有文献中对于 Storm 默认调度算法优化中尚存在问题，针对这些问题本文提出本文提出一种改进蚁群算法的 storm 任务调度优化算法。结合 storm 拓扑与蚁群算法，本文通过大量实验得出蚁群算法在 storm 任务调度中迭代次数、启发因子的最优取值范围，再引入 Sigmoid 函数对信息素挥发因子进行改进，信息素会随迭代次数的变化自适应调节，最后把各节点的资源（CPU、内存、磁盘等）与蚁群算法中信息素联系起来从而优化了 storm 任务调度。

1　Apache storm

1.1　Storm 作业模型

Storm 是 Twitter 开源的实时数据流计算系统，已经收录到 Apache 的孵化器中，由 Clojure 和 Java 语言开发。其借鉴了 Hadoop 的计算模型，Hadoop 运行的是一个 Job，而 Storm 运行的是一个 Topology，Job 是有生命周期的，而 Topology 是个 Service，并且是不会停止的 Job。图 1 展示了一个简单的 Storm 拓扑，包括一层输入 spout 和两层 bolt。为了保持性能，Storm 可以根据需要增加 blot 的并行性。拓扑是 Storm 的计算组织机制。它是用有向无环图（DAG）来表示的。众所周知，图是由顶点（节点）以边相互连接而成的结构。在这里，边是有向的，这意味着数据只能沿着边单向流动。这里的图是无环的，说明边的连接不会形成回路。否则，数据会在拓扑中无限流动。

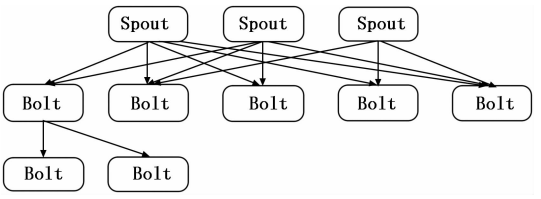

图 1　Storm 拓扑

1.2　Storm 默认调度策略

在 Storm 集群中，Storm 提交的拓扑作业按照总的任务数量平均分配到 Supervisor 的一作节点上，但是它忽略了不同节点的负载不同、性能不同，节点间的通信和进程间的通信问题。因此，Storm 默认调度可能导致高性能的工作节点的资源空闲，低性能的工作节点负载过重等问题以至于资源不能有效的利用。

2　标准蚁群算法

在蚁群算法中设蚂蚁的数量为 m，从第一个节点开始出发，每一只蚂蚁都根据转移概率来选择下一个节点，该节点到下一个节点的转移概率公式为：

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta})}{(\sum_{(s\in allowed_i)}(\tau_{is})^{\alpha}(\eta_{is})^{\beta}}, \text{若 } j\in \\ 0, otherswise\ allowed_i\end{cases} \tag{1}$$

式中，η_{is}为（i，j）相关的启发函数，τ_{is}表示节点i到节点j之间路径的信息素浓度，初始时刻各路径的信息素浓度相同。α为信息素浓度启发因子，代表信息素浓度的相对重要程度，该因子越大，蚂蚁越倾向于选择其它蚂蚁走过的路程；β为启发函数启发因子，表示启发函数的重要程度。

为了防止信息素无限制累加，引入信息素挥发机制，当蚂蚁完成一次从起点到终点搜索后，对每一条边上面的信息素浓度进行更新操作。具体为：

$$\begin{cases}\tau_{ij}(t+1)=\rho\tau_{ij}(t)+\Delta\tau_{ij}(t)\\ \Delta\tau_{ij}(t)=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t)\end{cases} \tag{2}$$

式中，$\Delta\tau_{ij}^{k}$（t）和 $\Delta\tau_{ij}$（t）为蚂蚁 k 和全部蚂蚁留在边（i，j）上信息素的增量；ρ 为信息素挥发系数。其中 $\Delta\tau_{ij}$（t）可按如下公式进行计算。

$$\Delta\tau_{ij}^{k}(t)=\begin{cases}\dfrac{Q}{L_k}, \text{若第 } k \text{ 只选择}(i,j)\\ 0, otherwise\end{cases} \tag{3}$$

式中，Q 为常数，表示信息强度；L_k 为第 k 只蚂蚁在本次迭代中走过的路径。

3　蚁群算法的改进

3.1　更改启发因子 α 与 β

启发因子反映蚂蚁在运动过程中所积累的信息量，指导蚁群搜索过程中的相对重要程度，其值越大，蚂蚁选择以前走过路径的可能性就越大，搜索的随机性减弱；而当启发因子值过小时，则易使蚂蚁的搜索过早限于局部最优[8]。通过设置更改启发因子，得出启发因子 α 与 β 在 storm 任务调度中的最佳取值范围，从而增加算法合理的搜

索指向性。实验结果分析具体在第 5 节有具体阐述，此处不再赘述。

3.2 挥发因子的 ρ 改进

信息素挥发因子 ρ 的的取值不仅影响到蚁群算法的搜索性能，而且不利于算法的收敛。ρ 性质为：ρ 过大时，信息素挥发快；反之，ρ 过小时，信息素挥发慢。在蚁群算法搜索的过程中，随着迭代次数的不断增加，信息素会在某些路径上累积，所以引入挥发因子 ρ 对其削弱。基于最初信息素累积不明显到迭代完成时累积过大的这种特性，我们引入 Sigmoid 函数，由于其单增以及反函数单增等性质，Sigmoid 函数常被用作神经网络的阈值函数，将变量映射到 0 到 1 之间。依 Sigmoid 函数这种性质我们将 Sigmoid 函数和信息素挥发因子 ρ 联系起来，蚁群算法的迭代次数为自变量，信息素挥发因子 ρ 为因变量。这样做刚好契合当蚁群算法在 storm 任务调度中的作用，在前期挥发因子对信息素的浓度削弱小，而到后期会增大对信息素的浓度削弱程度，随着迭代次数的改变挥发因子 ρ 自适应改变。信息素挥发因子 ρ 的表达式如（4）：

$$\rho = 1 + \frac{-1}{1+e^{\frac{x}{a}}},(0 < \rho < 1, b \neq 0) \tag{4}$$

式中，x 代表迭代次数，当 a 取迭代次数的二分之一时效果最好，b 可以控制 s 形曲线的胖瘦，引入变量 b 可以在工程当中去调节 b 的取值从而达到优化的目的。在第 5 节测出最优迭代次数，根据最优迭代次数计算出 b 的最优取值。信息素挥发因子 ρ 与迭代次数 x 的关系如图 2 所示。

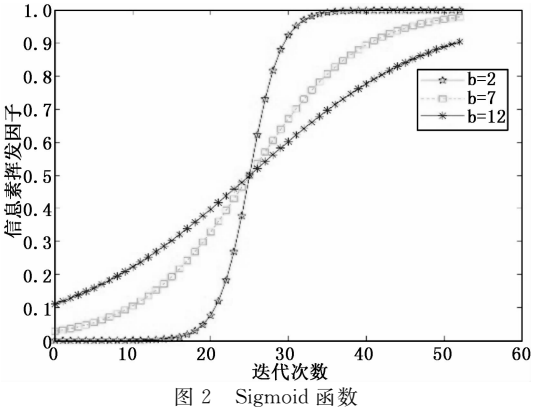

图 2 Sigmoid 函数

在第 5 节测出蚁群算法的最优迭代次数为 50 次左右，从而根据 Sigmoid 函数特性去设定（4）式中 a 取 25。实验尝试了 b 的取值对算法结果的影响，找到 b 的最优取值为 6 左右。

3.3 改进信息素浓度更新策略和负载均衡的优化

ZooKeeper 是 Google 开源的实现的分布式应用程序协调服务大数据组件，它可以为分布式应用提供一致性服务，包括：配置维护、域名服务、分布式同步、组服务等。Zookeeper 可以得到各个节点的心跳信息，Nimbus 周期性的利用 Zookeeper 上关于各个节点的可用资源情况。设节点上 CPU 信息素用 c 表示，内存信息素用 m 表示，磁盘信息素用 d 表示，网络信息素用 n 表示。为了防止超负载，把每一个参数的最大值设定为、、。此时节点的信息素为该节点的各类信息素加权和，分别用 e、f、g、h 代表 CPU、内存、磁盘、网络资源的权重，可以通过改变某类信息素的权重来增强或削弱该类资源对计算的影响。

$$\begin{cases} \tau = e\dfrac{c}{c_0} + f\dfrac{m}{m_0} + g\dfrac{d}{d_0} + h\dfrac{n}{n_0} \\ e + f + g + h = 1 \end{cases} \tag{5}$$

蚁群算法在计算过程中会使得某些节点资源累积过高（表现为信息素累积较高），这些节点会始终被分配任务；另外那些节点因为信息素浓度较低没有被分配任务从而造成负载不均衡。为此，引入控制负载因子 s，s_c 表示已经完成任务量，s_{sun} 表示任务的总量，τ_k 代表已经完成了 s_c 任务时的信息素，负载因子 s 通过优化了信息素浓度来防止负载不均衡。

$$\begin{cases} \tau = s \times \tau_k \\ s = s \ / \ s_{sun} \end{cases} \tag{6}$$

4 算法描述

4.1 算法预完成时间

在 Storm 任务调度中上，将 Topology 中的 n 个 tasks 线程分配到 m 个 Supervisors 工作节点上，有 $n>m$。设：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 线程；用 $SV=\{sv_1, sv_1, \cdots, sv_j\}$ 表示 m 个 Supervisor 节点集和，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，T_i 代表 i 线程所需资源（CPU，内存等）的多少，R_j 代表 j 节点当前所剩余的资源。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 $\boldsymbol{ET}[n, m]$ 可表示为：

$$\boldsymbol{ET} = \begin{bmatrix} et_{11} & et_{12} & \cdots & et_{1m} \\ et_{21} & et_{22} & \cdots & et_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ et_{n1} & et_{n2} & \cdots & et_{nm} \end{bmatrix} \tag{7}$$

式中，$et_{ij}=T_i/R_j$。

4.2 改进后蚁群算法的概率计算公式

蚁群算法中蚂蚁由节点 h 转移到节点 j 的状态转移概率公式如（8）：

$$p_{ij}^h = \begin{cases} \dfrac{(\tau_{ij})^\alpha (\eta_j)^\beta)}{(\sum_{j \notin tabu_h} (\tau_{ij})^\alpha (\eta_j)^\beta}, j \notin tabu_h \\ 0, otherswise, j \in tabu_h \end{cases} \tag{8}$$

式中，τ_{ij} 为从节点 i 到节点 j 的信息素；$\eta_j=1/et_i$，其中 et_i 为 t_i 任务的预期完成时间；α 和 β 为启发因子；$tabu_h$（$h=1, 2, \cdots m$）为不允许蚂蚁在走的节点，$tabu_h$ 会随着作业进行而不断的改变。

4.3 算法步骤

第一步：初始化所有参数；

第二步：设置最大迭代次数；

第三步：将任务分配到各个节点；

第四步：依据状态转移概率选择下一节点；

第五步：算法达到最大迭代次数输出结果，否则转移到第三步。

5　实验结果和分析

5.1　实验环境

集群由 8 台物理机器组成：master、slave1、slave2、slave3、slave4、slave5、slave6、slave7；每个节点配置 4 个 slot；每一个节点的资源配置都为 2.7 GHz CPU，4 GB 内存，2 TB 硬盘，10 Gbit 带宽的 LAN。各个节点 IP 等信息如图 3 所示。主服务进程部署在 master 节点，子服务进程都在其余 7 个 slave 节点上。经过调研和测试采用以下版本组合系统整体稳定性比较高，基础框架版本分别是：Centos：6.5、Hadoop：2.7.3、ZooKeeper：3.4.6、Hbase：1.2.4、Kafka：0.72、Storm：0.82、Storm－kafka：0.8.0－wip4。

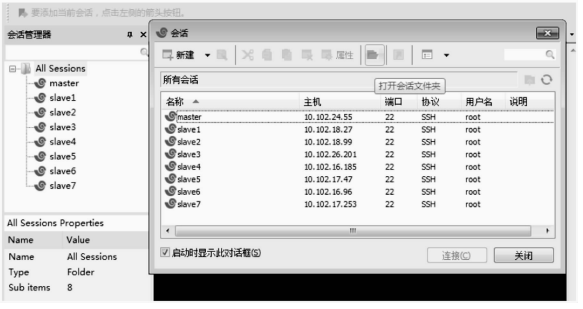

图 3　集群信息

本实验选择使用 RandomTextWriter[9] 生成随机单词文本作为测试数据（数据量大小可以人为改变），做 WordCount 单词统计任务。因为单词是随机生成的，所以各单词出现的频率不尽相同，因此在实际生产环境中具有一定的代表性。

5.2　通过测试寻找迭代次数

夏兰[8]在交通地理最佳路径的研究中中指出蚁群算法的最优在 65 次左右；侯守明等[10]在云任务调度优化中指出蚁群算法的最优在 60 次左右；尧海昌[11]在轨道交通集群调度问题中指出蚁群算法迭代次数最佳在 57 次左右。为了分析蚁群算法迭代次数与任务完成时间的关系。基于上述文献于此我们把迭代次数从 1 依次递增到 90，寻找最佳迭代次数。使用 RandomTextWriter 随机生成 300 MB 测试数据，对传统蚁群算法和改进的蚁群算法进行测试，测试结果如图 4、图 5 所示。

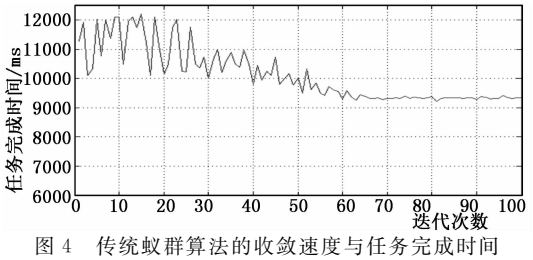

图 4　传统蚁群算法的收敛速度与任务完成时间

综合分析图 4 与图 5 可知，传统蚁群算法在迭代次数达到 62 次时开始收敛；改进蚁群算法在迭代次数达到 49 次时开始收敛；两者相比较改进蚁群算法收敛速度快。传统蚁群算法在收敛后任务完成时间大约为 9 300 ms；改进蚁群算法在收敛后任务完成时间大约为 7 300 ms；两者相比较改进蚁群算法完成任务时间快，大约比传统蚁群算法提高了 21.5%。

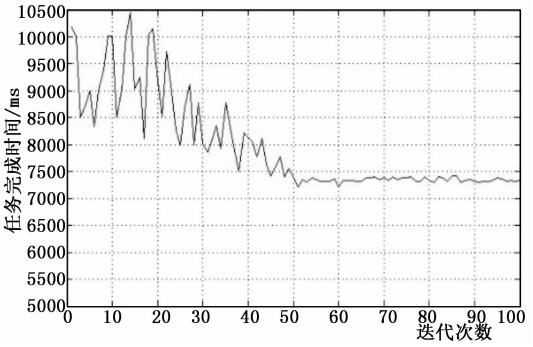

图 5　改进蚁群算法的收敛速度与任务完成时间

5.3　测试寻找启发因子 α 与 β 的最佳取值

简琤峰等[7]在 Storm 启发式均衡图划分调度优化问题中指出 α 的最佳范围在 0.5～2.5 之间，β 的最佳范围在 4～5.5 之间。尧海昌等[11]在基于蚁群算法的轨道交通集群调度算法研究中指出 α 的最佳范围在 3～5 之间，β 的最佳范围在 5～7 之间。张志文[12]在 AGV 路径规划与避障问题中指出 α 的最佳范围在 2～3 之间，β 的最佳范围在 2～4 之间。上述文献并没有指定 α 与 β 分别具体取什么值时效果最优，为此进行实验测试。使用 RandomTextWriter 随机生成 600 MB 测试数据，对改进的蚁群算法进行测试。首先进行粗粒度筛选，让 α 与 β 分别由 1 到 10 以 1 为间隔取值，计算出任务完成时间。统计结果得出当 $\alpha\in$［3，7］，$\beta\in$，［2，6］时任务完成时间最短。其次细粒度筛选，分别让 α、β 从以 0.1 为间隔从 $\alpha\in$［3，7］，$\beta\in$，［2，6］中取值计算出任务完成时间实验结果如图 6 所示。

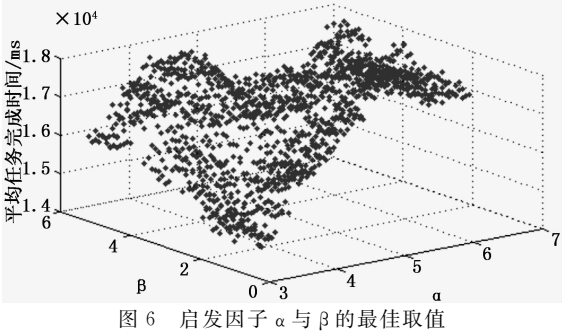

图 6　启发因子 α 与 β 的最佳取值

在图 6 中 x 轴代表启发因子 α，y 轴代表启发因子 β，z 轴代表平均任务完成时间。由图 6 可以看出，$\alpha=4$，$\beta\in$，［2，3，5］时算法最优，此时任务完成时间大约在 14 200 ms。当 α 与 β 分别取 6 和 5，7 和 6 左右时，任务完成时间对应为 15 500 ms 与 16 700 ms，此时由于 α 与 β 取值过大造成出现局部最优解；该问题与文献［13］在蚁群算法中参数 α、β、ρ 设置的研究中得到的结果不谋而合。由实验可知改进蚁群算法在 Storm 任务调度中 $\alpha=4$，$\beta\in$，［2，3，5］

时算法达到最优效果。

5.4 负载均衡对比

对于负载均衡我们来用对比测试各节点的 CPU 利用率来反映各个节点的负载情况。而 WordCount 业务逻辑处理随机单词的数据集可保证 Topology 上所有任务节点间的逻辑连线都有大量流事件经过，因此实验可客观展示出不同调度策略在流事件处理过程的各节点在实验当中 CPU 的使用情况。本实验使用 RandomTextWriter 随机生成 300 MB 测试数据，对 Storm 默认的轮询调度算法和改进的蚁群算法进行测试。图 7 为 Storm 默认的轮询调度算法和改进的蚁群算法在运行程序时集群中各个节点 CPU 的使用情况。

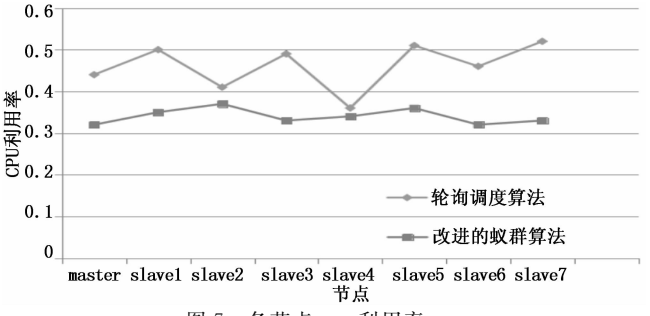

图 7 各节点 cpu 利用率

根据图 7 我们可以得到 Storm 默认的轮询调度算法在运行 WordCount 单词统计任务时，节点之间的 CPU 利用率的的标准差为 0.052，均值为 0.461。由标准差和均值可以看出 Storm 默认的轮询调度算法执行时，各工作节点的负载不均现象较为严重，其中 slave7 工作节点的 CPU 负载最高，slave4 工作节点的 CPU 负载最低，二者差值为 34%。执行改进的蚁群算法时，节点之间的 CPU 利用率的标准差仅为 0.017，均值为 0.342。从以上实验数据可以得出改进的蚁群算法和 Storm 默认轮的询调度算法相比 CPU 使用标准差降低了 3.5%，平均 CPU 负载降低了 26%。

5.5 任务完成时间分析

在试验中，使用 RandomTextWriter 随机生成测试数据，生成 200 MB、300 MB、400 MB、500 MB、600 MB 的随机单词文本数据集进行实验。默认的轮询调度算法和改进的蚁群算法相应规模大小的任务完成时间关系图如图 8 所示。

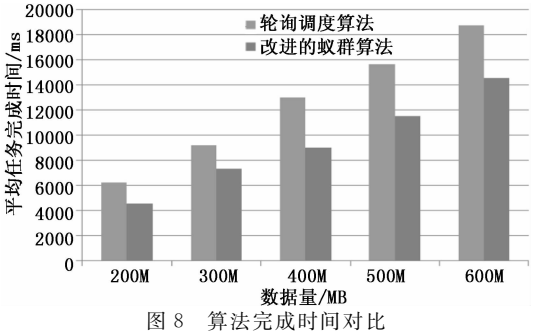

图 8 算法完成时间对比

由图 8 可以看出改进的蚁群算法比默认的轮询调度算法在任务完成时间方面降低了大约 21.6%，且随着任务量的增加，轮询调度算法完成时间与改进的蚁群算法差距越来越大。

6 结束语

本文针对 Storm 的默认任务调度机制中资源分配不合理而导致处理数据延迟问题，提出了改进蚁群算法在 Storm 任务调度中的优化方案。通过实验找出蚁群算法在 Storm 任务调度中的最佳迭代次数；测试得出启发因子 α 与 β 的最佳取值；引入 Sigmoid 函数改进了挥发因子 ρ，使其可以在算法运行时自适应的去调节。在文最后中将改进的蚁群算法和 Storm 默认的轮询调度算法做比较，得出改进的蚁群算法比 Storm 默认的轮询调度算法在平均 CPU 负载降低了 26%同时 CPU 使用标准差降低了 3.5%，在算法效率上 Storm 默认的轮询调度算法提高了 21.6%。得出改进蚁群算法的 Storm 任务调度明显优于 Storm 默认的任务调度。

参考文献：

[1] Peng B，Hosseini M，Hong Z，et al. R－Storm：resource－aware scheduling in Storm [A]. Middleware Conference [C]. ACM，2015. 149－161.

[2] Xu J，Chen Z，Tang J，et al. T－Storm：traffic－aware online scheduling in Storm [A]. 2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS) [C]. IEEE Computer Society，2014. 535－544.

[3] Re－Stream：real－time and energy－efficient resource scheduling in big data stream computing environments [J]. Information Sciences，2015，319：92－112.

[4] Ghaderi J，Shakkottai S，Srikant R. Scheduling Storms and Streams in the Cloud [A]. Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems－SIGMETRICS" 15 [C]. 2015：439－440.

[5] Eskandari L，Huang Z，Eyers D. P－Scheduler：adaptive hierarchical scheduling in apache Storm [A]. the Australasian Computer Science Week Multiconference [C]. ACM，2016.

[6] 蒋 溢，罗宇豪，朱恒伟. Storm 集群下一种基于 Topology 的任务调度策略 [J]. 计算机工程与应用，2018.

[7] 简琤峰，卢 涛，张美玉. Storm 启发式均衡图划分调度优化方法 [J]. 小型微型计算机系统，2018，39 (11).

[8] 夏 兰. 基于蚁群算法的交通地理最佳路径的研究 [D]. 武汉：武汉理工大学，2009.

[9] 蒋 溢，罗宇豪，朱恒伟. Storm 集群下一种基于 Topology 的任务调度策略 [J]. 计算机工程与应用，2018.

[10] 侯守明，张玉珍. 基于时间负载均衡蚁群算法的云任务调度优化 [J]. 测控技术，2018，37 (7)：31－35.

[11] 尧海昌. 基于蚁群算法的轨道交通集群调度算法研究 [J]. 南京邮电大学学报：自然科学版，2018，38 (4)：81－88.

[12] 张志文. AGV 路径规划与避障算法的研究 [D]. 成都：电子科技大学，2018.

[13] 叶志伟，郑肇葆. 蚁群算法中参数 α、β、ρ 设置的研究——以 TSP 问题为例 [J]. 武汉大学学报 (信息科学版)，2004，29 (7)：597－601.