计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190600048

面向边缘计算的Storm边缘节点调度优化方法

简琤峰　平　靖　张美玉
浙江工业大学计算机科学与技术学院　杭州 310023
(jiancf@zjut.edu.cn)

摘　要　边缘计算有高实时性和大数据交互处理的需求,边缘异构节点间的调度时耗长、通信时延高以及负载不均衡是影响边缘计算性能的核心问题,传统的云计算平台难以满足新的要求。文中研究了在边缘计算环境下Storm边缘节点的调度优化方法,建立了面向边缘计算的Storm任务卸载调度模型。针对拓扑任务在边缘异构节点间的实时动态分配问题,提出了一种启发式动态规划算法(Inspire Dynamic Programming,IDP),通过改变Storm的Task实例的排序分配方式以及Task实例和Slot任务槽的映射关系实现全局的优化调度;同时,针对拓扑任务的并发度受限于JVM栈深度的缺陷,提出了一种基于蝙蝠算法的调度策略。实验结果表明,与Storm调度算法相比,所提算法在边缘节点CPU利用率指标上平均提升了约60%,在集群的吞吐量指标上平均提升了约8.2%,因此能够满足边缘节点之间的高实时性处理要求。

关键词:边缘计算;Storm;资源调度;动态规划;蝙蝠算法
中图法分类号　TP391

Edge Computing-oriented Storm Edge Node Scheduling Optimization Method

JIAN Cheng-feng,PING Jing and ZHANG Mei-yu
College of Computer Science & Technology,Zhejiang University of Technology,Hangzhou 310023,China

Abstract　Edge computing has the demands of high real-time and big data interactive processing. The key problems of edge computing performance are long scheduling,high communication latency and unbalanced load among the edge heterogeneous nodes. Traditional cloud computing platforms are difficult to meet these new requirements. This paper focuses on the scheduling optimization method of Storm edge nodes in the edge computing environment. Firstly,a Storm task offloads scheduling model for edge computing is established. And then a heuristic dynamic programming algorithm is put forward to realize real-time dynamic allocation of topological tasks among edge heterogeneous nodes. By changing the sorting method of the Task instance and the mapping relationship between the Task instance and the Slot,the global optimization scheduling is achieved. Aiming at the problem that the concurrency of topological tasks may be greater than the maximum depth of the JVM stack,a scheduling strategy based on bat algorithm is put forward,the global scheduling scheme is calculated according to the information of the topology task and the CPU information of the edge node. Experiments show that compared with the current Storm scheduling algorithm,the proposed algorithm has an average increase of about 60% in the CPU utilization metrics of the edge node and an average increase of about 8.2% in the throughput metrics of the cluster. Therefore,the proposed algorithm can meet the high real-time processing requirements between edge nodes better.

Keywords　Edge computing,Storm,Scheduling,Dynamic planning,Bat algorithm

1　引言

为了减少由任务计算产生的能量消耗,将计算任务卸载到边缘服务器[1-2]。在边缘计算环境下,通过边缘设备对数据进行预处理,将计算任务卸载到多个边缘节点,能够降低带宽消耗和数据传输难度,大大提高了服务质量。但是将任务传输到边缘服务器会消耗额外的能量,完成卸载任务的时间也会增加[3],因此边缘节点的任务调度问题已成为当前研究的热门话题[4-6]。使用何种调度方式将任务分配给合适节点,使边缘节点间的调度时耗最短、通信代价最低、节点资源的利用最合理,同时保证集群的负载均衡是边缘节点调度问题的关键[7]。

到稿日期:2019-06-11　返修日期:2019-10-07　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金面上项目(61672461,61672463)
This work was supported by the National Natural Science Foundation of China (61672461,61672463).
通信作者:张美玉(zmy@zjut.edu.cn)

云计算平台的数据处理主要有批处理和流式处理两种方式,但只有以 Storm 为代表的实时流式处理平台符合边缘计算的应用需求。Storm 的 EvenScheduler 调度机制采用轮询方式来分配任务,在边缘环境下这种方式会导致边缘节点间的通信代价高、负载不均以及边缘节点资源利用率低等,严重影响边缘计算数据的传输性能。而 Storm 改进版本新增的 ResourceAwareScheduler 调度机制通过感知节点资源来优化通信代价和资源利用率,但并没有从本质上改变 Storm 的拓扑任务线程的 Task 实例的分配方式,在边缘环境下依然会造成调度时间过长、负载不均的问题。因此,现有的 Storm 调度机制并不能满足边缘计算的要求。

针对上述问题,本文研究了在边缘计算环境下 Storm 边缘节点的全局调度优化方法,建立了面向边缘计算的 Storm 调度模型,提出了一种启发式动态规划算法和一种基于蝙蝠算法的调度策略,并在此基础上研究了面向边缘计算的 Storm 调度优化方法。

2 相关工作

优化边缘计算的调度问题和实时流式处理框架的一个重要挑战是优化任务拓扑的部署[8]。最近,已有相关研究以最适合异构场景的方式扩展了 Storm 的调度方式,即设置了额外的输入(如网络链接信息或节点资源利用率等)和附加的模块(如附加的系统监视器或复杂的调度程序)。例如,文献[9-12]描述了 Storm 的扩展,并考虑了服务器之间的 CPU 和网络负载,以便重新平衡任务到节点的分配;文献[13]借助 Metis 工具对任务拓扑进行多层 K 划分;文献[14]提出利用 GPU 来提升 Storm 计算能力的方法。但是,文献[9]增加了额外资源感知检测模块来优化节点通信代价和资源利用率,从而增加了调度方法的复杂度;文献[11]使用贪心算法来检测网络延时,容易陷入局部最优。这些方法仅额外附加感知模块,却没有从本质上改变拓扑的分配方式,因此依然存在调度时间过长、负载不均的问题。如何降低 Storm 集群边缘服务器之间的调度时长以及通信代价,提高节点的负载均衡度和资源利用率是亟需解决的关键问题。目前也有相关研究使用动态规划编程来优化边缘计算环境下的调度问题[15]。此外针对任务调度,文献[16-19]采用智能搜索算法(粒子群、蚁群、遗传算法等)来优化 NP-Hard 问题,并取得了不错的效果。虽然粒子群等算法已被广泛应用于科学和工程领域,但在处理具有不同特征问题方面仍有所欠缺,而蝙蝠算法作为近年出现的智能算法之一,被证明在一些问题的处理上比 GA 和 PSO 的效果更好[20-22]。

本文提出了一种启发式动态规划算法,其改变 Storm 调度框架中 Task 实例的排序分配方式以及 Task 实例和 Slot 任务槽的映射关系,然后根据边缘节点配置检测的结果计算最优的全局调度方案。该算法的特点是在拓扑任务设置的并发度低于在 JVM 设置的最大栈深度,能准确计算出全局最优方案,其缺点是如果拓扑并发度高于阈值,会造成栈溢出。针对此问题,本文提出了一种基于蝙蝠算法的调度策略,结合 Storm 的调度问题根据 Task 实例和 Slot 任务槽的映射关系初始化一个随机解,通过蝙蝠算法的不断迭代来计算最优解。本文算法复杂度低、运行速度快,适合任何并发情况,且无须手动配置参数,能将属于同任务的线程最大化地分配到相同节点,保证了边缘节点的通信代价最低。

3 Storm 调度边缘节点的调度模型和优化方案

3.1 集群架构

边缘计算环境下的 Storm 集群由 Storm 云中心的 Master 单元节点和若干异构边缘服务器组成,如图 1 所示。

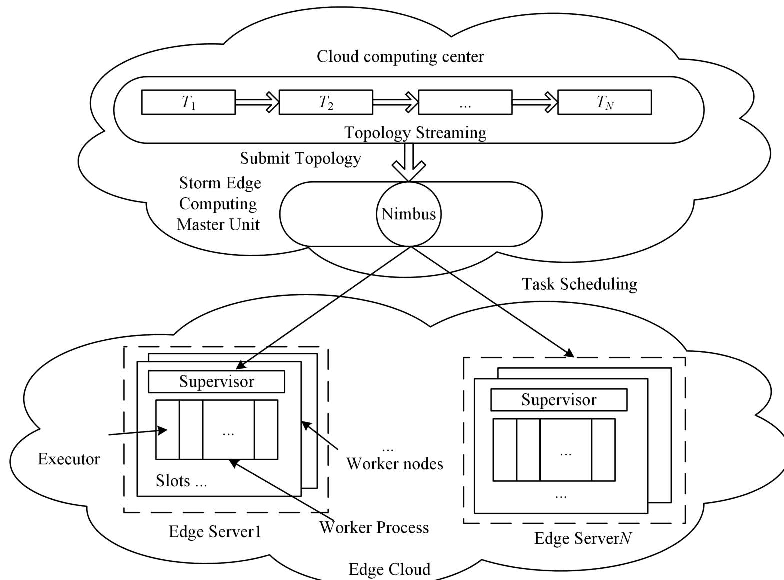

图 1 Storm 集群架构

Fig. 1 Cluster architecture of Storm

图 1 中,Storm Edge Computing Master Unit 为云中心与边缘服务器 Storm 集群节点间的通信 Master 单元节点,它管

理云中心拓扑流任务卸载到各边缘服务器的 Storm 集群节点的通信与调度。其装载的 Storm 的核心组件 Nimbus,作用是收集和管理云中心卸载到边缘服务器的拓扑数据流,然后根据相应的调度算法将其分配到各边缘服务器的子节点。子节点装载了 Supervisor 组件,Supervisor 有一个或多个 Worker 进程。Supervisor 将任务委派给 Worker 进程,Worker 进程根据需要产生尽可能多的 Executor 线程并运行拓扑任务。每个子节点运行在各个边缘服务器中,组成一个边缘云。

3.2 任务调度模型

Storm 集群中每个节点的 Supervisor 启动一个或多个 Worker 进程。所有的 Topology 都将在 Worker 进程里运行,进程启动的最大数量由该边缘节点配置的 Slot 决定。

现有的 Storm 调度模型的 Sort-slots 算法以轮询分配的方式,将拓扑任务中的 Executor 线程的 Task 实例以〈node id+port id,(*start-task-id*,*end-task-id*)〉集合的形式逐一分配给可用的 Slots。这样的方式只是简单的排序再分配,在拓扑任务数量多的情况下会造成节点的负载不均,且没有考虑节点的配置信息。

本文提出的调度模型改变了 Storm 的分配方式,抽象出一种组合优化的集合形式,具体以每个 Slots 被分配到 Executor 线程中 Task 实例的数量组成一个集合,再将该集合分配给可用的 Slots,集合的形式实际表示为一个一维数组。同时,在模型中附加一个获取边缘节点配置信息的模块。图 2 给出了 Storm 任务调度模型。

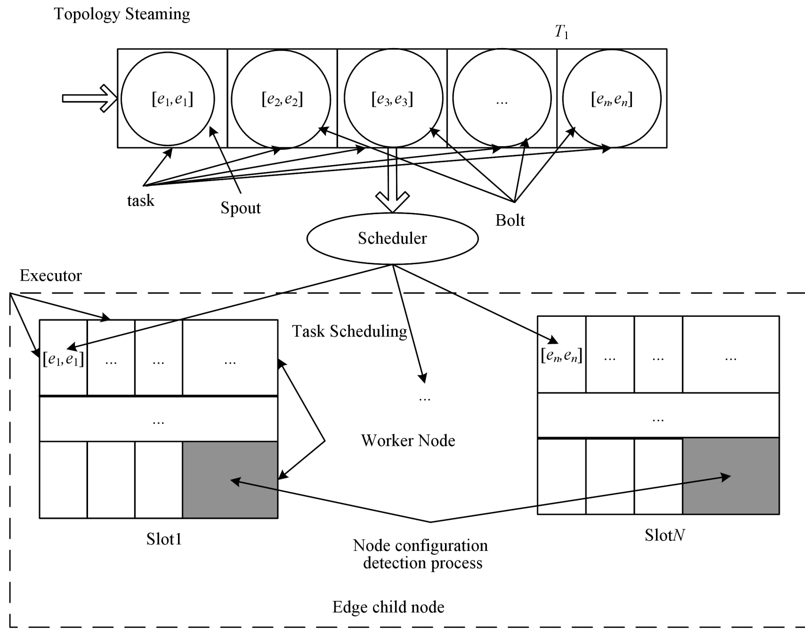

图 2 Storm 任务调度模型

Fig. 2 Task scheduling model of Storm

为了方便理解本文模型的调度过程,须对 Storm 集群的结构作如下定义。

对于一个含有 n 个工作节点 Storm 集群 $N=\{n_i \mid i\in[1,n]\}$,每个节点 n_i 配置有 S_i 个 Slot,那么集群可被分配的计算资源,即 Slot 集合 R 可表示为:

$$R=\{S_j^i=\langle i,j\rangle i\in[1,n],j\in[1,S_i]\} \tag{1}$$

其中,S_j^i 表示第 n_i 个节点的第 j 个 Slot。集群中共有 $Total_S(N)$ 个 Slot 资源。

对于即将提交至集群的 Topology T,一个任务的拓扑 T 由 Worker 进程中的 Executor 线程所组成,Executor 中的(Spout 或 Bolt)Task 实例是由开始和末尾的 *task -id* 组成的二维数组的序列,其单元格式可定义为:

$$[start\text{-}task\text{-}id,end\text{-}task\text{-}id] \tag{2}$$

一般情况下,Task 中开始和末尾的 *id* 是相同的,即 *start-task-id* = *end-task-id*。这里统一定义为 $E_i(i\in(1,N))$。

图 2 中,Scheduler 调度器以上述抽象出的集合的形式(集合中的每个维度为每个 Slot 被分配到 Executor 线程数量,集合的形式实际表示为一个一维数组)将拓扑数据流 $T1$ 中的 Task 实例通过调度器 Scheduler 分配到相应的边缘节点。分配模型如图 3 所示。

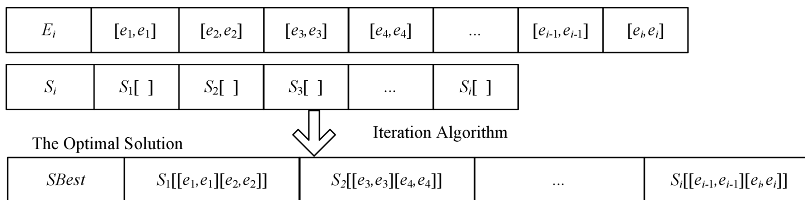

图 3 Scheduler 分配模型

Fig. 3 Scheduler assignment model

Task 实例的 Executor 线程以式(2)的形式均匀分配到相应节点所对应的 Slot 的空集合，最终每个 Slot 集合中存储的是被分配的 Executor 线程的数量，那么对 T 的资源调度可以表示为：

$$f(x)\rightarrow S \tag{3}$$

其中，函数 f 表示 Executor 到 Slot 的映射，x 表示执行 Task 的 Executor 集合和容纳 Executor 的集合，S 表示对应的 Slot。

3.3 优化方案

本文 Storm 的调度问题可以简化为：如何将 $N_e(T)$ 个 Executor 线程分配到 $N_S(T)$ 个 Slot 集合中，使得 Storm 在边缘节点的调度时间最短、资源利用率最高，同时保证节点间的负载均衡。这是一个多项式求解 NP-Hard 问题。

为了优化上述调度过程，本文将 $N_e(T)$ 个 Executor 分配到 $N_S(T)$ 个 Slot 的分配方案看作群组的一个解，解的数量则由 Topology T 所配置的 Executor 线程数和 Worker 进程数决定。每个解的结构是上述抽象出的集合的形式，所有解的解集 $res=\{res_1,\cdots,res_k\}$ 为 k 个解组成的二维数组。T 所配置的 Slot 数 $N_S(T)$ 和 Executor 数 $N_e(T)$ 被视为优化的维度。通过附加配置检测模块，将获取的边缘节点配置信息作为调度的输入，将一次任务调度的总执行时间和每个边缘节点的负载均衡标准差作为解的评价值，则利用启发式动态规划算法和基于蝙蝠算法的调度策略计算得到的最优解就是 Storm 调度的最佳分配方案。

4 算法设计

4.1 启发式动态规划算法

本文算法首先根据边缘节点配置检测的结果(CPU 的利用率)作为适应度函数来评价解的优劣。

定义 1 对于第 N_{res} 个分配方案，即解为 $res[i]$ $(i\in[1,N_{res}])$，提交至集群的 Topology T 所配置的 Executor 数为任务的长度 $L_{Ne(T)}$，获取系统分配给集群可使用的最大 CPU 利用率为 $C_{\rm Sys}$，分配给 Executor 的 CPU 占集群总值的百分比为 $P_{\rm Exe}$，则完成一次将既定数量的 Executor 线程分配到第 i 个 Slot 所需的调度执行时间 T_i 的计算为：

$$T_i=\sum_{i=1}^{N_S(T)}\frac{res[i]}{\left(C_{\rm Sys}\times\left(\frac{1}{res[i]\times P_{\rm Exe}}\right)\right)} \tag{4}$$

T_i 越小，说明该调度方案的整体执行时间越短。在计算每个解的最短执行时间的同时，需要考虑每个节点的负载均衡。负载均衡的标准差越小则负载越均衡。

定义负载均衡的标准差 LB 的计算公式为：

$$LB=\sqrt{\frac{1}{N_S(T)}\sum_{i=1}^{N_S(T)}(T_i-T_{\rm avg})^2} \tag{5}$$

启发动态规则算法的目的是将 $N_e(T)$ 个 Executor 分配到 $N_s(T)$ 个 Slot，保证解的总执行时间最短且负载均衡度最高。针对上节建立的调度模型，计算所有可能的分配结果。

启发式动态规划算法首先初始化解集 $res=\{res_1,\cdots,res_n\}$，定义当前第 i 个 Slot 的索引为 Idx，为充分利用本地资源，设定每个 Slot 允许容纳的最大、最小 Executor 数，分别定义为：$MaxN_e(T)$ 和 $MinN_s(T)$，其取值范围为 $(1,N_s(T)+1)$；然后按照图 3 的方式，根据式(4)、式(5)计算最优解。该算法的具体过程如算法 1 所示。

算法 1 启发式动态规划算法

Step1 输入 Topology T 配置的 Executor 的数量 $\rm N_e(T)$、Slot 的数量 $\rm N_s(T)$，以及 $\rm MaxN_e(T)$，$\rm MinN_e(T)$；

Step2 初始化当前已经分配的 Executor 线程数为 0，即 Currrent $\rm N_e(T)=0$；

Step3 判断当前节点的 Slot 的索引 Idx，如果当前索引值小于 $\rm N_s(T)$ 且当前已分配 Executor 线程数小于 $\rm N_e(T)$，则根据全局变量索引 Idx，循环遍历将第 j 个值赋值到 res[Idx]中；

Step4 重复 Step2 和 Step3，将还未分配的任务数补位到数组 res[$\rm N_s(T)$]的位置，得到当前节点的调度方案集合；

Step5 若还存在没有处理的节点，则以没有处理的节点为新的当前节点(使用递归的形式重复上述循环遍历过程)，返回 Step2，直至计算得到所有可能的调度方案集合；

Step6 对解进行评价，以得到的最优解作为 Storm 节点任务调度的最佳分配方案。

4.2 基于蝙蝠算法的调度策略

基于蝙蝠算法的调度策略首先根据图 3 的方式随机初始化一个群组解，再根据式(3)、式(4)不断计算迭代出全局最优解，然后使用基于出入栈的思想，根据最优解中的 Task 实例的数量，将 Executor 以(*start-task-id*, *end-task-id*)集合的形式分配到相应的 Slot 中，分配时将属于一个 Slot 的多个 Executor 线程分配在一起，以优化其通信代价高的问题。基于蝙蝠算法的调度策略的具体流程如算法 2 所示。

算法 2 基于蝙蝠算法的调度策略

Step1 遍历 Topology T，判断拓扑是否需要调度，否则结束算法。

Step2 获取 T 从 Component id 到 Executor id 的映射的 Map 集合，将获取的 Component 的 id 的 Map 集合存入 Set 集合；将 T 配置的 Executor 的 id 按 Component 的 id 的顺序存入 Collection 集合。

Step3 根据 T 配置的 Worker 数量确定所需要的 Slot 数量。

Step4 根据蝙蝠算法得出的解 res 中的分配方案将 Executor 分配到 Worker 中，将结果存入 List 集合。

Step5 获取集群可用的 Slot 并排序存入 List 集合；如果 Slot 已满则释放 Slot。

Step6 调用 Assign 方法将分配好的 Slot 集合分配到集群的边缘节点中，到此算法结束完成分配。

4.3 算法复杂度分析

本文提出的启发式动态规划算法在生成初始解时运用了递归的方式，在拓扑任务并发度较低的情况下能精准计算全局最优的调度方案，其时间复杂度是 $O((Max-Min)^M)$，其中 Max 和 Min 是每个 Slot 分配的 Executor 数量的范围，M 是 Slot 的数量，算法执行的最差情况是 $n=M*Max$。拓扑任务设置的并发度大小受限于 JVM 栈深度，所以启发式动态规划算法只适用于拓扑任务并发度较低的情况。基于蝙蝠算法的调度策略是不确定算法，首先随机产生初始解再根据初始解不断迭代寻优计算得到相对最优解，其精度不如启发式动态规划算法，时间复杂度为 $O(N*M)$，其中 N 是蝙蝠的数量，M 是最大迭代次数，最好情况下的时间复杂度为 $O(N)$。

由此可知,蝙蝠算法的时间复杂度要明显优于启发式动态规划算法指数级的复杂度,且不受限于 JVM 的栈深度,但该算法的缺点是可能得到全局相对最优解。

5 实验与分析

5.1 实验环境

本文实验由安装 ESXI6.0 系统的 Dell R710 服务器上的虚拟化集群来模拟边缘服务器之间的交互,并模拟出 4 个不同配置的节点。服务器的基本配置如下:Intel(R) Xeon(R) X5650 的 CPU,2.66 GHz×6 core×2,128 GB 的 RAM,1 Gbps×4 的网卡以及 2 个 1T 硬盘组成的磁盘阵列。虚拟节点的配置如下:第 1 虚拟节点的 CPU 为 2.67 GHz×1 core,RAM 为 2 GB,操作系统是 Ubuntu 16.04 x86_64;第 2 个虚拟节点的 CPU 为 2.67 GHz×2 core,RAM 为 4 GB,操作系统是 Ubuntu 14.04 x86_64;第 3 个虚拟节点的 CPU 为 2.67 GHz×4 core,RAM 为 6 GB,操作系统是 Ubuntu 14.10 x86_64;第 4 个虚拟节点的 CPU 为 2.67 GHz×6 core,RAM 为 8 GB。操作系统是 Ubuntu 12.04 x86_64。

集群配置如表 1 所列。

表 1 主机配置

Table 1 Storm node configuration

主机	IP	功能
Storm-M	192.168.0.18	Nimbus,Zookeeper
Storm-S1	192.168.0.19	Supervisor,Zookeeper
Storm-S2	192.168.0.20	Supervisor,Zookeeper
Storm-S3	192.168.0.21	Supervisor,Zookeeper

本文使用为 1.06 Storm 版本的,利用 Pluggable Scheduler 实现所提算法。实验中使用 Word Count Topology 来测试集群性能。

5.2 调度算法对比

本节对比了蝙蝠算法、PSO、GA 以及本文提出的启发式动态规划算法(IDP)。在 Storm 环境下,并发度(拓扑任务设置的 Worker 和 Executor 数量)不同时,调度方案的计算时间的差异如图 4 所示。

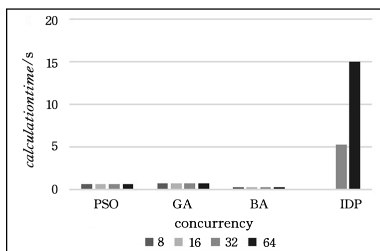

图 4 不同并发度下计算时间的对比

Fig. 4 Comparison of calculation time under different degrees of concurrency

启发式动态算法的特点是并发度低时拥有最快的计算时间和计算全局最优解的能力,符合边缘计算的强实时性需求。但是由于其时间复杂度较高,随着并发数增加,计算时间会呈指数增长,在边缘计算环境下便不再适用。而智能算法先随机产生初始解,再根据初始解不断迭代寻优计算得到相对最优解,其复杂度较低且受并发数的影响较小。从图 4 可以看出,IDP 算法在并发度较小的情况下的计算速度最快,但是当并发度设置为 32 或 64 时,计算时间呈指数增加。智能算法受并发度影响较小,且蝙蝠算法的计算时间较 PSO 和 GA 更短,更符合 Storm 在边缘计算的应用场景。

5.3 调度模型评价指标

边缘环境下的高实时处理需求是 Storm 边缘节点调度优化问题的关键。集群吞吐量是影响集群实时处理性能的一个关键指标,它指单位时间内处理的数据量。其中,决定集群吞吐量的指标有 CPU 使用率和负载均衡等。本文使用以下指标来评估算法性能。

(1)单位时间元组(Tuple)的吞吐量(Tuple/s),即每秒处理的 Tuple 的数量,其直接反映集群的吞吐量。在 Storm 中,一个数据流由无数个 Tuple 序列组成,这些元组会被分布式并发地创建和处理。单位时间内集群处理的元组数能直接反映集群处理的数据流,即吞吐量的大小。吞吐量越大,集群处理数据的效率越高。

(2)指定时间内集群中所有边缘节点的平均 CPU 使用率。该指标的数值越大,每个节点的 CPU 资源利用率就越高,即越能充分利用节点的计算能力。

(3)指定时间内集群中所有边缘节点平均 CPU 使用量的标准差。该指标的值越小,说明每个节点的 CPU 利用率的差异越小,即节点负载越均衡。

5.4 对比实验

本实验中的数据来源于 Storm 提供的 Trident RAS API 和 Storm UI REST API,以及本文算法实现的边缘节点配置检测模块。实验的比较对象为:1)Storm,Storm 的默认均衡调度模型;2)E-Storm,本文实现的面向边缘的 Storm 模型;3)R-Storm,文献[10]提出的资源感知的实时 Storm 模型。

5.4.1 边缘节点的 CPU 使用情况

表 2 列出了 3 个调度模型的调度场景中,拓扑任务启动后 1min 内每 5s 记录的所有边缘节点的 CPU 使用情况的平均值。该值的单位是拓扑任务进程 CPU 使用率的百分比。

表 2 调度开始 1 min 内节点间平均 CPU 使用率的对比

Table 2 Comparison of average CPU usage among nodes within 1 minute from start of scheduling

Timestamp/s	CPU 使用率/%		
	Storm	R-Storm	E-Storm
1	4.55	10.85	9.4
5	4.3	11.55	7.3
10	4.75	10.15	6.7
15	4.7	7.7	5.5
20	3.9	8.45	8.3
25	4.5	7.1	7
30	4.55	8.2	6.45
35	4.25	6.55	10.35
40	4.15	7.7	5.4
45	3.8	10.7	6.2
50	5.4	7.6	6.3
55	4.75	5.75	6.55
60	4.5	6.9	7.55

为了更直观地展示这 3 种调度模型的优化效果比较,本

文使用线形图来表示,如图 5 所示。Storm 默认的均衡调度模型采用单一的轮询方式来分配 Executor 线程,而没有考虑每个边缘节点的 CPU 负载情况,其特点是调度效率高但节点的 CPU 利用率低,会造成一定的资源浪费。R-Storm 需要手动设置各个组件运行时每个实例所需要的资源数和 Topology 优先级数。R-Storm 对 CPU 的负载进行了详细优化,因此其资源利用率高但调度效率低,且集群的负载性能较依赖手动设置的参数。本文实现的 E-Storm 可根据动态获取的 CPU 信息计算一个全局负载最优的调度方案,同时优化了节点资源利用率和调度效率。如图 5 所示,E-Storm 的优化效果明显优于 Storm,其在某些时刻的 CPU 利用率高于 R-Storm。虽然 R-Storm 的整体 CPU 利用率略高于 E-Storm,但差距并不明显。

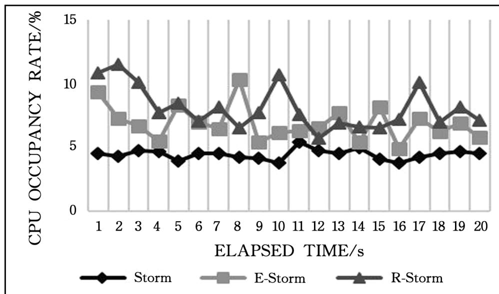

图 5 节点间 CPU 平均占用率

Fig. 5 Average CPU usage among nodes

5.4.2 边缘节点负载均衡度的对比

图 6 给出了 3 个调度模型的调度场景中每个节点 CPU 使用情况的标准差。标准差越小,意味着负载越平衡。由于默认的 Storm 采用轮询机制,且每次轮询后会对节点剩余的 Slot 进行排序,因此能保证负载的相对均衡。R-Storm 着重优化了节点资源的利用率,但由于需要手动设置参数,可能导致边缘节点的负载倾斜。E-Storm 是在资源利用率、调度效率、负载均衡中取一个相对平衡的策略。从图 6 可以看出,在目前的实验环境中,Storm 的负载平衡程度是最好的,E-Storm 的负载平衡度明显优于 R-Storm。由此可见,R-Storm 在负载均衡方面存在不足。

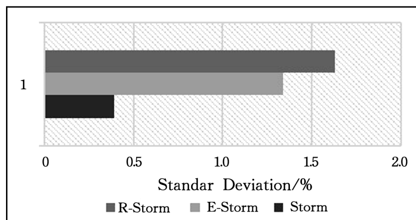

图 6 CPU 平均占用率的标准差

Fig. 6 Standard deviation of CPU average occupancy

5.4.3 集群吞吐量的对比

集群吞吐量是上述各项指标的综合评价结果,是影响边缘环境下实时数据处理的关键因素。吞吐量越大,集群在单位时间内的数据处理能力就越强。表 3 列出了调度开始后 1 min 内每 5 s 收集一次的每个调度模型的集群吞吐量数据,其表示每秒处理的元组数(Tuple/s)。

表 3 调度开始 1 min 内各调度器集群吞吐量的对比

Table 3 Comparison of cluster throughput under each scheduler within 1 minute from start of scheduling

Timestamp/s	吞吐量/(Tuple/s)		
	Storm	R-Storm	E-Storm
5	220	600	420
10	1060	1420	1520
15	2200	2300	2420
20	3100	3480	3380
25	4040	4300	4540
30	5000	5380	5380
35	5980	6360	6380
40	6880	7240	7380
45	7880	8400	8240
50	8760	9260	9420
55	9700	10100	10180
60	10780	11240	12240
65	12780	12100	13120
70	13080	13800	14160

为了更直观地展示集群吞吐量之间的差距,图 7 给出了调度启动后 1 min 内每 5 s 收集一次的每个调度程序下的集群吞吐量。集群的吞吐量受节点 CPU 利用率、负载均衡度、调度处理效率等指标的综合影响,调度算法能综合考虑这些指标并更好地平衡它们,从而有效地优化集群的吞吐量。默认的 Storm 只考虑了调度的效率和负载均衡,忽视了节点 CPU 利用率;R-Storm 着重优化节点的利用率,在负载均衡度和调度效率上略显不足。从图 7 可以看出,前期 E-Storm 的集群 Tuple 处理量与 R-Storm 的集群 Tuple 处理量相差不大,而前期 Storm 的集群吞吐量明显低于两个集群吞吐量,但在后期,直到系统最终稳定,E-Storm 的吞吐量高于其他两种模型。

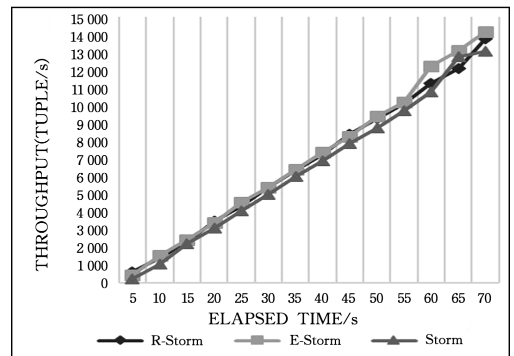

图 7 单位时间的 Tuple 发送量

Fig. 7 Tuple throughput in unit time

5.5 实验分析总结

综合上述实验,本文实现的 E-Storm 在各项指标的优化上均取得了较好的平衡。在节点 CPU 利用率和负载均衡指标的对比上,虽然 E-Storm 在负载均衡方面不如默认的 Storm,但是默认的 Storm 调度模型的 CPU 利用率较低,会影响集群的吞吐量;此外,R-Storm 侧重优化 CPU 利用率而在负载均衡的优化上略显不足,而 E-Storm 在负载均衡的优化上比 R-Storm 更出色。因此,本文实现的 E-Storm 模型在资源利用和负载均衡的整体优化上要优于其他两种模型,在最重要的集群吞吐量的指标上优化效果最好,更符合边缘节点的调度需求。

结束语 本文研究了边缘计算环境中 Storm 边缘节点的

计算卸载策略,提出了一种启发式动态规划算法,该算法能计算出满足条件的所有分配方案,并准确找到全局最优解。针对拓扑任务的并发度可能超过 JVM 设置的栈最大深度的问题,本文提出一种基于蝙蝠算法的调度策略,通过随机产生初始解,不断迭代计算出优化解。本文所提方法可以应用于最常见的场景,包括高拓扑并发性的情况,并且不需要手动配置参数。实验证明,对于边缘节点的调度优化,与当前的 Storm 调度算法相比,本文所提算法在边缘节点 CPU 利用率指标上平均提升了约 60%,在集群的吞吐量指标上平均提升了约 8.2%,因而能够满足边缘节点之间的高实时性处理要求,可以有效提高边缘环境中的数据传输能力。

参 考 文 献

[1] GUO H,LIU J,ZHANG J,et al. Mobile-edge computation offloading for ultradense IoT networks[J]. IEEE Internet of Things Journal,2018,5(6):4977-4988.

[2] PHAM Q,LE L B,CHUNG S,et al. Mobile edge computing with wireless backhaul:Joint task offloading and resource allocation[J]. IEEE Access,2019,7:16444-16459.

[3] ZHANG Y,CHEN X,CHEN Y,et al. Cost Efficient Scheduling for Delay-Sensitive Tasks in Edge Computing System[C]// Proceedings of 2018 IEEE International Conference on Services Computing. 2018:73-80.

[4] KIM Y,KWAK J,CHONG S. Dual-Side Optimization for Cost-Delay Tradeoff in Mobile Edge Computing[J]. IEEE Transactions on Vehicular Technology,2017,PP(99):1-1.

[5] ZENG D,GU L,GUO S,et al. Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System[J]. IEEE Transactions on Computers,2016,65(12):1-1.

[6] GU L,ZENG D,GUO S,et al. Cost Efficient Resource Management in Fog Computing Supported Medical Cyber-Physical System[J]. IEEE Transactions on Emerging Topics in Computing,2017,5(1):108-119.

[7] JIAN C F,CHEN J W,PING J,et al. An Improved Chaotic Bat Swarm Scheduling Learning Model on Edge Computing[J]. IEEE Access,2019,7(1):58602-58610.

[8] CHENG B. Edge-Computing-Aware Deployment of Stream Processing Tasks Based on Topology-External Information:Model,Algorithms,and a Storm-Based Prototype[C]// IEEE International Congress on Big Data. IEEE,2016.

[9] PENG B,HOSSEINI M,HONG Z,et al. R-Storm:Resource-Aware Scheduling in Storm[C]// Middleware Conference. ACM,2015.

[10] ANIELLO L,BALDONI R,QUERZONI L. Adaptive Online Scheduling in Storm[C]// Proceedings of the 7th ACM International Conference on Distributed Event-based Systems. ACM,2013:207-218.

[11] CARDELLINI V,GRASSI V,PRESTI F,et al. Distributed QoS-aware Scheduling in Storm[C]// ACM International Conference on Distributed Event-Based Systems. ACM:2015:344-267.

[12] JIAN C F,LU T,ZHANG M Y. Storm Scheduling Optimization Method Based on Graph Partitioning Strategy[J]. Journal of Chinese Computer Systems,2018,39(11):2538-2544.

[13] ESKANDARI L,HUANG Z,EYERS D. P-Scheduler:adaptive hierarchical scheduling in apache storm[C]// Proceedings of the Australasian Computer Science Week Multiconference. ACM,2016.

[14] CHEN Z H,XU J L,TANG J,et al. G-Storm:GPU-enabled High-throughput Online Data Processing in Storm[C]// 2015 IEEE International Conference on Big Data. IEEE,2015:307-312.

[15] ZHANG W,HU Y,HE H,et al. Linear and dynamic programming algorithms for real-time task scheduling with task duplication[J]. The Journal of Supercomputing,2017,75(2):494-509.

[16] XIE Y,ZHU Y,WANG Y,et al. A novel directional and non-local-convergent particle swarm opti-mization based workflow scheduling in cloud – edge environment[J]. Future Generation Computer Systems,2019,97:361-378.

[17] MOON Y J,YU H C,GIL J M,et al. A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments[J]. Human-centric Computing and Information Sciences,2017,7(1):28.

[18] DENG X H,GUAN P Y,WAN Z W,et al. Integrated Trust Based Resource Cooperation in Edge Computing[J]. Journal of Computer Research and Development,2018,55(3):449-477.

[19] FU X. Task Scheduling Scheme Based on Sharing Mechanism and Swarm Intelligence Optimization Algorithm in Cloud Computing[J]. Journal of Computer Science,2018,45(6):290-294.

[20] KONGKAEW W. Bat algorithm in discrete optimization:A review of recent applications[J]. Songklanakarin Journal of Science and Technology(SJST),2017,39(5):641-650.

[21] JIAN C F,CHEN J W,PING J,et al. An Improved Chaotic Bat Swarm Scheduling Learning Model on Edge Computing[J]. IEEE Access,2019,7(1):58602-58610.

[22] JIAN C F,LI M,QIU K Y,et al. An improved NBA-based STEP design intention feature recognition[J]. Future Generation Computer Systems,2018,88(6):357-362.

JIAN Cheng-feng, born in 1973, Ph.D, postgraduate, associate professor, is a member of China Computer Federation. His main research inte- rests include cloud computing, CAD and image processing.

ZHANG Mei-yu, born in 1965, M. D, professor, is a member of China Computer Federation. Her main research interests include data mining and image processing.