吴宗泽, 陈庆奎. 面向 AI 数据流的基于负载阈值的双模态负载均衡算法[J]. 智能计算机与应用,2025,15(12):1-8. DOI: 10.20169/j.issn.2095-2163.251201

面向 AI 数据流的基于负载阈值的双模态负载均衡算法

吴宗泽, 陈庆奎
(上海理工大学 光电信息与计算机工程学院, 上海 200093)

摘 要: 为解决 AI 并发数据流中计算节点负载失衡问题,本文利用数据流的周期性特征,提出了一种基于负载阈值的双模态负载均衡算法。在低负载场景下,采用基于优劣解距离法(Technique for Order Perference by similarity to Ideal Solution, TOPSIS)的静态分配算法,通过整合 CPU 利用率、内存占用率、网络带宽等关键指标,量化计算节点的静态性能,并依据此量化结果进行数据流分配,同时还确定了触发动态策略的负载转换阈值;在高负载场景下,启用基于改进遗传算法的动态优化算法:通过引入迭代状态自适应的概率操作机制,动态调整交叉概率与变异概率,构建了以计算节点集群的负载均衡度和平均总时延为优化目标的效用函数模型。实验结果表明,基于负载阈值的双模态负载均衡算法显著提升了计算节点间的负载均衡性,并优化了系统整体性能。

关键词: AI 数据流; 负载均衡; 静态性能; 遗传算法; 双模态算法
中图分类号: TP368.5 **文献标志码**: A **文章编号**: 2095-2163(2025)12-0001-08

A dual-modal load balancing algorithm based on load thresholds for AI data streams

WU Zongze, CHEN Qingkui
(School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: To address the problem of load imbalance among computing nodes in AI concurrent data streams, this paper proposes a dual-modal load balancing algorithm based on load thresholds, utilizing the periodic characteristics of data streams. The core design of this algorithm is as follows: in low-load scenarios, a static allocation algorithm based on TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) is employed. This means integrating key indicators such as CPU utilization, memory usage, and network bandwidth to quantify the static performance of computing nodes, and data stream allocation is conducted based on this quantification. Thresholds for triggering dynamic strategies are also identified. In high-load scenarios, a dynamic optimization algorithm based on an improved genetic algorithm is activated. By introducing an iterative state adaptive probability operation mechanism, the crossover probability and mutation probability are dynamically adjusted, creating a utility function model with load balancing degree of computing node clusters and average total delay as optimization objectives. Experimental results indicate that the dual-modal load balancing algorithm based on load thresholds significantly enhances the load balancing among computing nodes and optimizes the overall system performance.

Key words: AI data flow; load balancing; static performance; genetic algorithm; dual-modal algorithm

0 引 言

人工智能物联网(AIoT)是近几年来被广泛关注的人工智能(AI)和物联网(IoT)的融合状态,即人工智能算法技术与物联网在实际应用场景中的落地融合[1]。传感器设备会持续产生海量、实时、异构且规模不断增长的传感数据,并将其传输至计算工作端,构成 AIoT 系统处理与分析的基础[2]。面对此类数据带来的吞吐与时效挑战,实现高效可靠的 AIoT 系统尤其依赖于数据管理系统在实时处理、可

基金项目: 国家自然科学基金(61572325);上海重点科技攻关项目(19DZ1208903)。

作者简介: 吴宗泽(1999—),男,硕士,主要研究方向:网络计算与并行计算,边缘计算,数据流处理技术。
通信作者: 陈庆奎(1966—)男,博士,教授,博士生导师,CCF 会员,主要研究方向:计算机集群,并行计算,人工智能等。Email:chenqingkui@usst.edu.cn。

收稿日期: 2024-03-07

靠存储与高效转发方面的性能[3-4]。边缘计算和云计算是 AI 和物联网结合场景中非常重要的两个关键使能技术[5]。无论是边缘计算还是云计算场景下,数据流的构建和数据流向的管理是搭建高性能应用场景的关键[6]。

在数据流系统中,大量的新数据不断地产生和流入,而新增的数据又必须经过实时处理[7]。得益于其强大的并行计算能力、流水线处理模式以及对函数式编程的支持,数据流编程模型已被众多主流计算系统广泛采用[8]。Kafka 作为 LinkedIn 开发的分布式流处理平台,最初服务于其内部的数据管道和活动流处理[9-10]。历经多年发展,Kafka 已成为一个成熟的平台,能够在多种应用场景下高效、实时地处理海量数据流,并广泛应用于不同领域,验证了 Kafka 在工业环境中的高性能与可靠性[11-12]。然而,现有研究表明 Kafka 在应对 AI 数据流场景时存在局限性,针对 Kafka 中间件在处理 AI 数据流时面临的实时性挑战,需要解决 AI 并发数据流中计算节点负载失衡问题[13]。

计算节点间的负载不均衡是制约分布式架构整体性能提升的常见瓶颈。在以往的研究中,AI 数据流的负载均衡问题的解决方案主要有两种思路,第一种思路使用静态算法分析节点的性能。静态算法在系统结构较为简单和计算性能宽裕时能有一定的作用,而在复杂的负载调整中局限性显著,难以应对负载波动、缺乏对运行时节点状态的感知能力;第二种思路是根据实时性能情况动态变更策略,但是动态算法也存在一些问题,例如在节点负载较低时均衡效率下降、算法对总体性能的消耗较大,易出现少量峰值点使得短期内数据分配不均等。

文献[14]提出了一种自适应负载指标权值的负载均衡算法,在对服务器节点分类的基础上,利用节点性能和综合负载信息分配任务,从而避免负载倾斜问题,但该算法仅在相似配置的服务器构建的环境下进行测试,无法展现高复杂度环境下的负载均衡状态;文献[15]提出了一种基于动态反馈的加权最小连接数服务器负载均衡算法,根据计算节点在运行中负载动态的变化和负载权重的更迭变化来完成资源的合理输送和负载的均衡,但在低负载下分配效率较低;文献[16]设计一种计算理想贴近度作为动态权重的 TOPSIS 模型,结合熵值赋权法计算负载信息指标的权重对后端服务器进行综合权重赋予,最大化利用集群资源,但无法在多变的高负载环境下完成负载均衡;文献[17]提出了基于模拟退火策略与自适应权重策略的优化方法,旨在提高粒子群调度算法的自我优化能力,但出现了低负载下能耗较高的问题。

为了解决现有方法存在的问题,本文将静态量化算法和改进的遗传算法结合应用于 AI 数据流的负载均衡处理,增加算法对数据流操纵的准确性,利用改进遗传算法寻找 AI 数据流分配的合理方案,通过实时的遗传迭代算法实现 AI 数据流负载均衡控制。

1 系统模型及实现

1.1 AI 数据流定义

目前流行的对源数据流进行预处理的方式是通过传感器等终端设备先对源数据流进行处理。本文将这种经过预处理的带有 AI 源数据特征的数据流定义为 AI 数据流。经过 AI 预处理后的数据,即 AI 数据流,都具有一定的特征,最主要的特性是数据的矩阵化,这表明大多数 AI 数据的存储格式都是浮点型数据的阵列,使得传输中的 AI 数据都具有相对固定的模式和标准;AI 数据流的另一个特征是数据的产生和发送存在周期性。传感器等终端以周期的方式产生着 AI 源数据,也以周期的方式向处理链条的下游传递 AI 数据流。在 AI 数据流单元的构建中要考虑数据流的周期性质,从而使处理 AI 数据流的流程都能呈现更好的健壮性和高效性。

所以,针对 AI 数据流的特点,本文定义了 AI 数据流的基本格式:(AIStreamId, Ctime, AIData, Size, PeriodTime)。每一个发出的 AI 数据流都具有自身唯一的流 AIStreamId;Ctime 是该数据流的产生时间,多条由终端处理并生成的数据流会按照时间序列发送;AIData 保存 AI 数据部分;Size 表示 AI 数据流的大小;PeriodTime 标识该 AI 数据流的生成周期,告诉接收方需要在一定的时间周期内处理完该数据流。

依据以上 AI 数据流结构,本文对数据流量进行了最小划分,在计算分配方案时利于算法流程的优化,为 AI 数据的整体流通、识别和存储奠定了基础。

计算节点在高负载的复杂情况下的计算能力偏向于黑盒,所以需要使用动态调度算法实时生成负载分配方案。本文提出的算法将算法模式切换(静态/动态)与动态算法的负载再分配操作均锚定在周期性框架下执行,算法模式切换时机严格限定在规划周期的边界点进行。为了避免过于频繁的切换而导致的资源消耗,本文将算法切换的周期定义为规划周期。规划周期一般由多个小周期组成,使用

规划周期在策略合理变动和保持资源稳定之间找到平衡点，可以为分配策略的计算工作提供更多信息。

1.2　系统架构

系统整体架构设计如图 1 所示。整体架构分为五大模块：传感器数据采集模块、AI 数据抓取模块、系统控制模块、AI 数据计算模块以及数据存储模块。传感器数据采集模块主要是接入边缘终端设备的生产数据，这些数据具有数据量大、实时性以及周期性的特点，对采集系统有高并发、实时处理数据的要求；AI 数据抓取模块主要使用 Kafka 平台，不断将 AI 数据流推送到 Kafka 的信息队列中，Kafka 的消息系统集群包含了多个负责管理和存储消息的组件（broker），使流式数据可以得到稳定的获取和转发；AI 数据计算模块由多个数据分析节点组成；数据计算模块同样以周期的形式来计算 AI 数据；数据控制模块管理整体 AI 数据流的流向与配置，通过配置中心可以管理不同模块中的节点的配置信息；监控中心则用于获取抓取节点当前的运行状态和在最近周期中的量化效率信息和负载情况；数据存储模块是由关系型数据库 Mysql 和非关系型数据库 Redis 共同搭建的，使用不同的数据库可以保存系统中不同性质的数据。

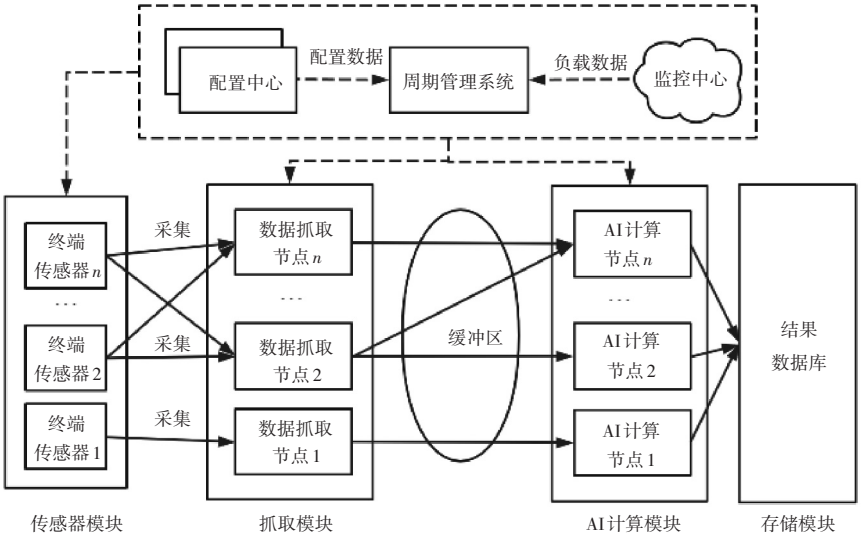

图 1　系统架构

Fig. 1　System architecture

通过配置中心向系统中配置抓取、计算、采集节点的信息，传感器节点对 AI 数据进行预处理后，将 AI 数据流推送到 Kafka 集群，抓取节点通过配置信息得知当前节点所需抓取的 AI 数据流信息，在采集了 AI 数据流后，依据配置信息表把 AI 数据流推送到相应的 AI 计算节点，计算结束后会将结果入库。

1.3　基于阈值的整体算法

本文提出了一种基于阈值的动静态结合的双模态负载均衡算法，整体算法流程如图 2 所示。根据静态量化算法的最终指标特点配置了阈值，建立搭载了监控模块的负载均衡管理系统；当监控到计算节点集群的负载在阈值以内时，采用基于 TOPSIS 的静态分配算法；当负载在阈值之上时，将数据流分配策略动态调整为以计算节点集群的负载均衡度、平均总时延作为效用函数的遗传算法。

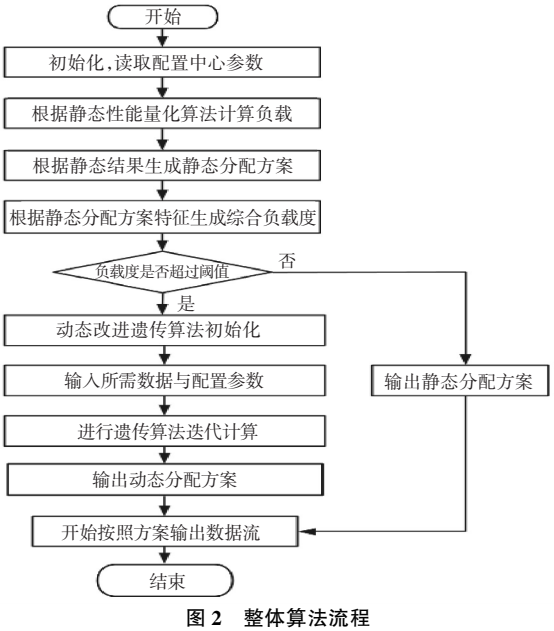

图 2　整体算法流程

Fig. 2　Overall algorithm flow

在低负载的情况下,使用动态算法会使系统能耗过高,也易出现负载均衡的短暂波动,造成资源浪费[18]。在负载较小的情况下,只需要收集计算节点的性能信息并使用基于权重的量化方法测算节点性能,使用静态性能计算数据流的分配工作。本文将低负载情况下的基于权重的量化和分配算法定义为静态分配方法,其输出为静态分配结果。在高负载的情况下,确定的权值对于不同场景下的计算往往不具有适配性,对于可能需要经过不同计算模型的 AI 数据流来说,确定的比重权值并不能很好的针对复杂的计算环境,但同时对负载情况存在影响。因此,本文在复杂的高负载场景下,通过改进的遗传算法来完成负载均衡任务。

1.4 计算节点的静态性能量化

静态负载均衡分配需要先获取到每个计算节点的性能或状态。每个计算节点的性能由多种参数决定,在一般的计算场景下,CPU 利用率、内存利用率、磁盘利用率、网络带宽等是衡量一个计算节点性能的常规参数。为了更好地应对 AI 数据流计算场景,还需增添 GPU 利用率。

本文将通过 TOPSIS 计算得到节点的理想贴近度作为其性能权重值,该权重值的取值范围在 0 到 1 之间,用于衡量数据流分配方案与理想性能目标的接近程度,性能权重值越与 1 相近,则相应的目标结果更倾向于最优水平;反之,越接近 0,则对应的目标结果更接近差水平。与普通的权值计算方法相比,TOPSIS 更能综合的考虑各个参数,避免单个参数影响过大的情况出现[19]。在负载均衡过程中,系统根据各计算节点的权重,分配合理比例的数据流供计算节点计算。本文用 U_i 表示各个计算节点的利用率,$i(i=1,2,3,\cdots,n)$ 表示第 i 个计算节点,U_i_cpu,U_i_mem,U_i_disk,U_i_net,U_i_gpu 分别代表第 i 个计算节点的 CPU 利用率、内存利用率、磁盘利用率、网络利用率和 GPU 利用率。

(1)建立计算节点各指标的正向化矩阵 $\boldsymbol{E}$,r_{ij} 是第 i 个计算节点的第 j 个参数,正向化矩阵如下式:

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

(2)对各利用率参数进行标准化处理,得到标准化的矩阵,正向化矩阵中的每一项都经由下式计算:

$$Z_{ij}=\frac{r_{ij}}{\sqrt{\sum_{i=1}^{n} r_{ij}^2}} \tag{2}$$

(3)得到标准化的参数矩阵 $\boldsymbol{Z}$:

$$\boldsymbol{Z}=\begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{21} & z_{22} & \cdots & z_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nm} \end{bmatrix} \tag{3}$$

(4)对参数矩阵 $\boldsymbol{Z}$ 的每一行分别计算各项负载信息参数的最大值和最小值,下标 i 的取值范围为 $i(i=1,2,3,\cdots,n)$,代表每个计算节点的数据情况:

$$\boldsymbol{Z}_i^{+}=\max(z_{i1},z_{i2},\cdots,z_{im}) \tag{4}$$

$$Z_i^{-}=\min(z_{i1},z_{i2},\cdots,z_{im}) \tag{5}$$

(5)计算每个计算节点的负载信息参数和其最大值、最小值之间的欧式距离:

$$D_i^{+}=\sqrt{\sum_{j=1}^{m}\omega_j(Z_j^{+}-Z_{ij})^2} \tag{6}$$

$$D_i^{-}=\sqrt{\sum_{j=1}^{m}\omega_j(Z_j^{-}-Z_{ij})^2} \tag{7}$$

其中,ω 是用户根据算法特性和实践测试情况定义的每个使用率的权重,权重越高,利用率对节点静态性能的影响就越大。

(6)由式(6)和式(7)得到评价指标,也即静态量化性能指标:

$$L_i=\frac{D_i^{-}}{D_i^{-}+D_i^{+}} \tag{8}$$

(7)根据评价指标,获取该节点在整体系统中的综合静态负载情况,周期管理中心输出静态数据流分配方案,使用 C_i 来表示综合负载情况:

$$C_i=\frac{L_i}{\sum_{i=1}^{n} L_i} \tag{9}$$

(8)周期管理中心依据比例化的数据流分配表开始分配,比例化的数据流分配向量:

$$\boldsymbol{A}=[NC_1,NC_2,NC_3,\cdots,NC_n] \tag{10}$$

其中,N 是数据流总数,C_i 表示数据流分配结果。

上述工作构建了静态量化算法,并且同时计算了由静态算法适宜区间转变为动态算法适宜区间的并发 AI 数据流的关键阈值。静态算法可以很好地反应标准情况下节点的性能状况,本文通过对静态算法的结果的延申利用和监控中心得到的相关周期信息,计算出动静态算法转换测算公式:

$$S = \frac{\sum_{i=1}^{n} \frac{o_i}{NC_i}}{n} \times 100\% \tag{11}$$

其中，o_i 是当前计算节点在该规划周期内实际处理的数据流数量，N 是该规划周期内总 AI 数据流量。

S 值越大，说明系统整体的运算效率越高。随着负载率的上升，S 值逐渐下降，说明静态算法逐渐无法输出合理的方案。当阈值设置为 80% 左右且 S 处于阈值之上时数据处理效率以及计算节点负载状况最好。当 S 低于阈值时，说明当前 AI 数据流的计算效率偏低，需要更新分配方案，即开始采用动态算法。

1.5　改进的负载均衡遗传算法

本文采用噪声过滤机制，在负载的周期变化下，如果波动较小，则不会触发数据流的重新分配。本文使用负载周期变化率 ΔL 来分析波动情况，决定下一个周期是否需要重新上传当前计算节点负载信息，计算新的数据流分配方案：

$$\Delta L = \left| \frac{L_1 - L_2}{t_1 - t_2} \right| \tag{12}$$

其中，ΔL 表示负周期变化率；L_1、L_2 表示两个不同时刻 t_1、t_2 对应的负载值；t_1、t_2 表示两个不同的时间节点。

配置负载变化感知值 Δd，当本周期收集到的负载变化率小于该感知值时，说明此次负载变化程度较小，不会引起该节点数据流的再分配，也不会上传本周期负载信息。

遗传算法通过模仿自然选择和生物遗传进化过程来得到问题最佳解的方法。进化的重要驱动因素是交叉、重组或杂交。而变异操作则通过引入随机性变化维持种群多样性，对探索解空间至关重要[20]。因为 AI 数据流的产生存在周期性，本文在动态分配过程中还将考虑到周期之间的计算延迟时间。针对 AI 数据负载均衡处理的问题，本文设计了基于改进遗传算法的动态优化算法作为动态优化策略的核心。

本文设计一种改进遗传算法的参数编码方法。将每个 AI 计算节点分配的 AI 数据流的量化数量方案作为遗传算法初始种群中的染色体 $X = (x_1, x_2, \cdots, x_n)$，$x_i$ 表示第 i 个计算节点在下一个周期的数据流分配数量，n 为参与 AI 数据流分配的节点数量。总的 AI 数据流数量 N 满足下式：

$$\sum_{i=1}^{n} x_i = N \tag{13}$$

种群的初始化采用完全随机或有限制的随机生成方式，需要在解空间内产生一定数量的随机解，然后不断对其优化以完成解的全局搜索。为了使初始种群更加有利于对 AI 数据流分配方案的搜索，本文中采用少量染色体特殊生成和剩下染色体随机生成的方式初始化种群，随机生成的染色体在一定程度上维持了搜索空间的全局性，特殊染色体则为了在种群内添加寻找最优方案的有利信息，可以有效加快最终解的寻找。

第一个特殊染色体为 AI 数据流平均分配方案，将所有数据流等量地交给所有计算节点；第二个特殊染色体为静态性能预计分配方案，根据每个节点通过静态测试算法得到的量化性能对数据流进行分配；最后一个特殊染色体考虑到前一个计算周期的分配信息和负载情况，，通过搭载上一周期的信息可以有效提升最优解的搜索效率。

适应度是为了判断当前解是否适宜生存，也就是是否更优。适应度函数需要计算当前染色体所代表的染色体方案是否能够使所有计算节点在整体上获取更低的计算延迟。生成分配方案的周期是依赖传感器的传输周期的，所以在传输周期的基础上需要考虑到计算节点周期的差异性。因此适应度函数需要基于周期性延迟情况，首先计算第 i 个计算节点处理单位量 AI 数据流的预计t_i：

$$t_i = \frac{\sum t_f}{x_i} \tag{14}$$

其中，t_f 是对于每一个计算节点上一个传输周期内，所有计算周期完成计算所分配的 AI 数据流的时间。

如果无计算延迟，完成时长就是计算时长；如果有计算延迟，完成时长是周期时长加上延迟时长。

其次，进一步计算第 i 个节点在当前染色体所代表的数据流分配方案下的预计处理时长 T_i：

$$T_i = \mathrm{e}^{x_i \times t_i} \tag{15}$$

计算得到预计的总执行时长和平均执行时长：

$$T_{\text{total}} = \sum_{i=1}^{n} T_i \tag{16}$$

$$\bar{T} = \frac{T_{\text{total}}}{n} \tag{17}$$

为量化负载分配的预期均衡程度，本文采用样本标准差作为评估指标。样本标准差提供了衡量数据量分配离散程度的客观度量基准。具体而言，计

算整个分配方案中各计算节点实际负载的标准差，该值能有效反映负载分布的均衡性 BL：

$$\mathrm{BL}=\sqrt{\mu\left(\frac{\sum_{i=1}^{n}(T_i-\bar{T})^2}{n}\right)} \tag{18}$$

其中，μ 是相关系数，$0<\mu\leqslant 1$。

适应度函数需要综合考虑计算总时长和每个节点的延迟情况，如下式所示：

$$f(x)=\frac{n}{\mathrm{BL}+T_{\mathrm{total}}} \tag{19}$$

采用适宜的选择策略可以在算法启动前期保护适应度较低的染色体不被淘汰，防止算法的演化走向固化，导入了特殊染色体等特殊处理，迭代不同环节的影响程度不同，用来调整迭代前期和后期的选择策略。适应度函数处理后的选择概率如下式：

$$P_s(x_i)=\frac{\exp(\frac{f(x_i)}{\theta})}{\sum_{j=0}^{n}\exp(\frac{f(x_j)}{\theta})} \tag{20}$$

其中，$P_s(x_i)$ 为第 i 条染色体的选择概率；$f(x_i)$ 为第 i 条染色体的适应度值；θ 是迭代中概率的控制参数；n 为种群中染色体总体。

随着迭代的进行，θ 逐渐变大。对适应度较高的染色体，被选择的概率较高，适应度较低的染色体则相反。选择概率在一开始更偏向于均分，而在后期则会放大原本的适应度影响。

对于交叉操作来说，先交换两条染色体的数据。随机挑选两个染色体，在第一条染色体中寻找两个基因 x_a 和 x_b，在第二条染色体中寻找两个基因 x_c 和 x_d，使得 $x_a+x_b=x_c+x_d$。如果找到了，交换对应基因；如果没有寻找到，则重复搜寻一定次数。在查找一定次数后还未找到合适的基因时则采用自交叉策略。预配置交叉操作发生的概率，若随机概率超过该概率，则该轮要执行交叉操作。

变异操作与标准算法相近，随机对某个基因变异，并另找一个基因对其进行配平操作，通过计算当前迭代次数中适应度来决定变异操作的概率。首先，在已经完成适应度计算后，得到当前迭代轮次的变异概率 P_v：

$$P_v=\frac{\rho e^{-k_1 g-k_2(\frac{f_{\max}}{f_{\mathrm{avg}}}-1)}\sqrt{n}}{Y} \tag{21}$$

其中，$f_{\max}$ 是适应度的最大值；f_{avg} 是适应度的平均值；Y 是种群规模；k_1、k_2 和 ρ 是常数；g 是迭代轮数。

$f_{\max}$ 和 f_{avg} 的比值偏大时，表明当前种群的染色体适应度值分布较为分散，此时降低了变异算子操作的概率，使得适应度更高的染色体更容易被保留，同时加快了收敛的速度。

迭代次数达到最大，输出适应度最高的染色体，即最终分配方案。

2 实验测试及评估

为了验证基于负载阈值的双模态负载均衡算法的性能，以公交车站实时摄像视频为数据基础，分别准备 8 个计算节点机器，将 AI 数据流以某种算法分配到 8 个节点上，并在部署配置一致的 Kafka 客户端程序。测试时将 8 个节点匹配启动，同时向 Kafka 服务端以周期的形式发送 AI 数据流。Kafka 数据抓取节点接收到周期数据流之后，会对数据进行分析、预处理和分配方案计算，然后分发数据流到计算节点。整个过程由配置中心和周期管理中心共同管理。

2.1 实验环境

服务器环境与配置见表 1。

表 1 服务器环境与配置信息

Table 1 Server environment and configuration information

参数	属性
CPU	Intel Core i7-4700CPU@3.60 GHz
内存	Kingston 12 GB 1 600 MHZ
硬盘	Scagatc 500 GB 7200 r/min
Jdk 版本	JRE(build 1.8.0_151-b12)
Zookeeper 版本	Zookeeper-3.4.6
Kafka 版本	Kafka_2.11-2.0.0

根据硬件资源能力设置动态服务配置信息见表 2。

表 2 动态服务配置信息

Table 2 Dynamic service configuration information

配置	详细信息
Kafka 集群服务器配置	采用 3 台机器为一个集群计算节点，端口 9092，topic 为 cal-data
周期管理系统信息	处理周期为：60 s，数据类型为：公交车站实时图片
监控中心信息	发送服务心跳模块、发送数据处理能力的模块

2.2 实验结果和分析

为了测试基于负载阈值的双模态负载均衡算法的性能，以下实验均把加权概率择优算法和粒子群

算法作为对比算法。

实验一：实验 10 轮，每次逐渐加大数据流的输入量，结果取平均值。将结果中的单个周期图片处理数量进行比较，然后对比单个周期所有节点完成计算任务后的周期剩余时间，生成对比图，并发流处理效率对比图如图 3 所示，周期剩余时间对比图如图 4 所示。

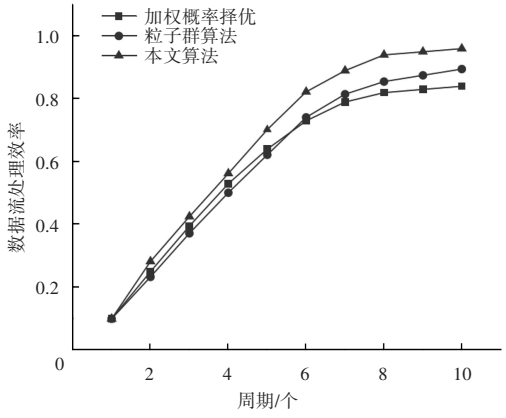

图 3　并发流处理效率对比图

Fig. 3　Comparison chart of concurrent stream processing efficiency

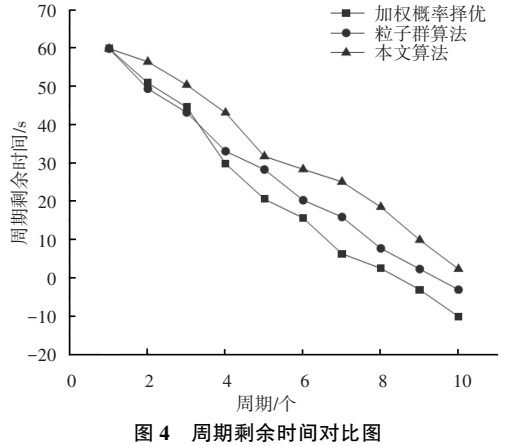

图 4　周期剩余时间对比图

Fig. 4　Comparison chart of the remaining time of the cycle

由图 3 和图 4 可知，相比于加权概率择优算法和粒子群算法，本文提出的基于负载阈值的双模态负载均衡算法随着周期数演进，单个周期内数据流量逐渐增大，均衡后资源的合理利用更加明显，在高输入流量的情况下数据处理效率依然较高。加权概率择优算法于流量较低的情况下效率较高，而在流量增加时表现变弱，粒子群算法于低流量区间表现较差。与粒子群算法相比较，本文算法在效率上具有 10%左右的提升。在低负载的情况下，本文算法能更好地判断流量的变化并作出合理的决策，降低系统能耗；在高负载的情况下，本文算法也能利用好所有服务器的性能且保证数据流分配的合理性，且在任务数量较大时改进效果越明显，说明改进算法能更好地应对大规模数据流的分配问题。

实验二：数据源头分别以每周期高数量、中数量、低数量 3 种传输模式交替，依次向数据抓取节点推送 AI 数据流，使用 8 个计算节点来测试数据处理情况，如图 5 所示。

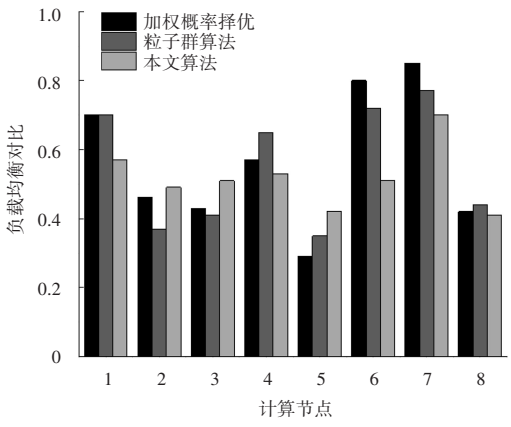

图 5　负载均衡对比

Fig. 5　Comparison of load balancing

从图 5 可以看出，如果对节点进行了不合理的数据流量分配，会导致节点性能使用的失衡。在周期数据流量频繁变化的情况下，静态算法无法很好的应对高负载下的均衡任务，综合的负载效果较差，出现了不同计算节点间负载差距较大的情况；粒子群算法出现了局部负载较高的情况；本文算法基于阈值动态切换算法，其节点性能使用更加平均与合理，负载均衡的性能更好。

综上，本文提出的基于负载阈值的双模态负载均衡算法既拥有静态算法在低负载环境下效率高、能耗低的优势，也在高负载环境下实现了更加良好的负载均衡效果，充分发挥了各计算节点的性能，使得整体系统架构的数据处理能力和压力承载能力都得到了加强，本文算法相比于静态加权概率择优算法、粒子群算法具有更好的负载均衡效果。

3　结束语

本文提出一种 AI 数据流分配与配置算法，从传感器设备接收数据并将数据收集到用于抓取数据的服务器，通过自定义的并发数据流结构和模块体系完成 AI 数据流的传输。针对物联网链路负载均衡控制效果差，系统整体资源浪费，且负载均衡性能差会导致计算节点性能使用不当，易发生数据发送接收错误的问题，结合静态负载均衡算法及动态负载均衡算法优点，有效提升了链路资源利用率，并显著降低了数据收发错误率。为验证本文算法的正确性，对加权概率择优算法、粒子群算法和本文算法进

行了对比实验,证明了本文算法在 AI 数据流负载均衡方面具有高效性。

参考文献

[1] 陆启镐. 基于异构计算的云边端融合 AIoT 架构设计及应用[D]. 杭州:杭州电子科技大学, 2023. DOI:10.27075/d.cnki.ghzdc.2023.000219

[2] 郑增乾, 王锟, 赵涛,等. 带宽和时延受限的流媒体服务器集群负载均衡机制[J]. 计算机科学, 2021, 48(6):261-267.

[3] 黄东生, 陈庆奎. 一个并发 AI 数据流处理节点内的通信模型[J]. 智能计算机与应用, 2022, 12(11):26-33.

[4] 彭世明, 林士飏, 贾硕,等. 基于负载预测的多目标优化任务卸载策略[J]. 计算机工程, 2024, 50(1):206-215. DOI:10.19678/j.issn.1000-3428.0066766

[5] 翁杰, 林兵, 陈星. 基于博弈论的多边缘服务器负载均衡策略[J]. 计算机科学, 2023, 50(S2):754-761.

[6] 刘欣, 李向东, 耿立校, 等. 工业互联网环境下的工业大数据采集与应用[J]. 物联网技术, 2021, 11(8):62-65. DOI:10.16667/j.issn.2095-1302.2021.08.020

[7] 陈庆奎, 吕晓明, 郝聚涛,等. 一个物联网异构数据接入系统 ChukwaX[J]. 计算机工程, 2012, 38(17):12-15.

[8] KAUSHIK M, JHARASHREE P, SANTOSH M K. A dynamic load scheduling in IaaS cloud using binary JAYA algorithm[J]. Journal of King Saud University - Computer and Information Sciences, 2020, 34(8):1043-1052.

[9] RAPTIS P T, CICCONETTI C, PASSARELLA A. Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications[J]. Future Generation Computer Systems, 2024, 154:173-188.

[10] RAPTIS T P, CLAUDIO C, MANOLIS F, et al. Engineering resource-efficient data management for smart cities with Apache Kafka[J]. Future Internet, 2023, 15(2):43.

[11] 金双喜, 李永, 吴骅, 等. 基于 Kafka 消息队列的新一代分布式电量采集方法研究[J]. 智慧电力, 2018, 46(2):77-82. DOI:10.3969/j.issn.1673-7598.2018.02.013

[12] 庞子皓. 基于 Netty 与 Kafka 的物联网数据采集平台的设计与实现[D]. 南京:南京邮电大学, 2023. DOI:10.27251/d.cnki.gnjdc.2022.000197

[13] 黄娅婷. 面向 Kafka 消息系统的性能优化方法研究[D]. 桂林:桂林电子科技大学, 2023. DOI:10.27049/d.cnki.ggldc.2023.000565

[14] 曲志坚, 张先伟, 曹雁锋,等. 基于自适应机制的遗传算法研究[J]. 计算机应用研究, 2015, 32(11):3222-3225.

[15] 张慧芳. 基于动态反馈的加权最小连接数服务器负载均衡算法研究[D]. 上海:华东理工大学, 2013.

[16] 张卓, 张上, 项天旭,等. 基于改进 TOPSIS 的动态权重负载均衡算法[J]. 计算机工程与设计, 2023, 44(11):3222-3229. DOI:10.16208/j.issn1000-7024.2023.11.004

[17] 马钰, 杨迪, 王鹏. 基于改进粒子群算法的云计算调度策略[J]. 长春理工大学学报(自然科学版), 2022, 45(5):80-86.

[18] 胡逸飞, 包梓群, 包晓安. 一种基于改进遗传算法的动静态负载均衡算法[J]. 电子科技, 2023, 36(9):79-85. DOI:10.16180/j.cnki.issn1007-7820.2023.09.012

[19] 钟立俊, 李武松, 李娅,等. 基于 EW 和 TOPSIS 的分布式模型调度技术[J]. 电子信息对抗技术, 2020, 35(6):78-82.

[20] FIGUEROA R O, CASTELLANOS Q M. An experimental approach to designing grouping genetic algorithms[J]. Swarm and Evolutionary Computation, 2024, 86:101490. DOI:10.1016/J.SWEVO.2024.101490