文章编号：2095-2163(2024)02-0106-08 中图分类号：TP391;TP311 文献标志码：A

一个并发数据流接入与动态配置模型

童 林，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：在万物互联的环境下，越来越多的设备接入网络，导致实时性并发数据流激增，对于网络传输、后台分析和存储有着重大考验。为解决传统数据接入与分析的低效率问题，以周期数据接入的方式，使用自定义协议、优化线程池、循环队列周期读写算法（CRW），提高系统的接入性能，增加系统的稳定性；为解决传统数据分析系统的配置问题，根据计算分析系统平均周期数据处理的效率，动态配置数据分析系统的节点。实验表明，数据接入模型在处理高并发请求方面，有较高的处理性能，且时延较低；数据分析系统使用 CRW 算法和动态配置策略后，数据处理速度显著提高，稳定性增强。

关键词：并发数据流；周期数据接入；动态配置；循环队列

A concurrent data stream access and dynamic configuration model

TONG Lin, CHEN Qingkui
(College of Optoelectronic Information and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: In the context of the Internet of Things, more and more devices are connected to the network, which has led to a surge in real-time concurrent data streams. The massive explosion of data poses a significant challenge to network transmission, backend analysis, and storage. To solve the inefficiency problem of traditional data access and analysis, a periodic data access method is adopted, which uses custom protocols, optimized thread pools, and cyclic cache queue cycle read and write algorithms (CRW) to improve the system´s concurrency ability, reduce data access latency, and thereby increase system stability. To solve the configuration problem of traditional data analysis systems, the nodes of the data analysis system are dynamically configured based on the efficiency of calculating the average cycle data processing of the analysis system. The experiment shows that the data access model has high processing performance and low latency in handling high concurrency requests. After using the CRW algorithm and dynamic configuration strategy, the data analysis system significantly improves data processing speed and stability.

Key words: Concurrent data flow; periodic data access; dynamic configuration; cyclic queue

0 引 言

工业与物联网设备的工作是长久、实时的，成千上万台的设备产生的数据流通过网络传输到后台，这些海量数据的采集、分析、存储对于生产调整与优化有着重要作用，通过分析数据的变化，可以展示生产的状况。工业实时性数据流具有规模大、速度快、类型杂等特点，数据的处理就需要一个支持高并发、低时延的数据接入系统和数据分析系统。

当前，针对工业、物联网并发数据流的接入与配置的研究中，越来越多的技术应用到数据接入系统的开发中。Netty 是一款高性能的通信框架，支持高并发连接、实时通信，文献[1]基于 Netty 通信框架实现智慧井盖监控系统对井盖的智能监控，实时获取井盖的监控数据。虽然使用 Netty 通信框架可以缓解高并发场景下终端设备的连接、传输压力，但是随着海量数据的同时采集，大量的同步操作极其容易造成程序的卡顿，从而造成较大的时间延迟、较低的吞吐量。Kafka 是一种消息队列，已经被大量应用于生产平台中，文献[2]将 Kafka 消息队列应用到电量采集系统

基金项目：国家自然科学基金(61572325)；上海重点科技攻关项目(16DZ1203603,19DZ1208903)；上海智能家居大规模物联共性技术工程中心项目(GCZX14014)；上海市一流学科建设项目(XTKX2012)；上海市重点项目(9DZ1208903)。

作者简介：童 林(1997-)，男，硕士研究生，主要研究方向：网络计算与并行计算、边缘计算及数据流处理技术。

通讯作者：陈庆奎(1968-)，男，博士，教授，博士生导师，CCF 会员，主要研究方向：计算机集群、人工智能、物联网大规模数据分析等。Email：chenqingkui@ usst.edu.cn

收稿日期：2023-02-24

中，实现电量数据实时汇集与分析；文献[3]把 Kafka 作为车辆套牌缓存中间件，设计出车辆套牌识别与分析系统，实现对车辆套牌事件的监测。

目前，数据接入技术已经应用在各个领域。文献[4]基于 REST(Representational State Transfer)技术设计一种物联网接入架构，使得系统之间结构更加的灵活，但 REST 的接口模式基于 HTTP 协议，在高并发场景下，海量并发数据流的同时接入，会造成较大的数据传输时延；文献[5]设计的流式处理，没有充分利用 CPU 资源，使用原始线程处理业务，增加数据处理的时间，在数据传输的过程中没有根据数据的类型自定义协议，增加传输的时延。在服务节点动态配置策略研究方面，文献[6]通过动态矩阵控制算法，将控制增量作为服务节点增量，但没有明确何时配置节点；文献[7]使用循环队列作为通信缓冲区，提高了数据通信的稳定性，但其对于数据是单批缓存，不便于数据的规模计算。

针对上述情况，在数据接入方面，为了解决高并发环境下数据接入过慢的问题，本文自定义通信协议，降低数据传输过程中的损耗，减少传输时延迟；优化线程池技术，设置合理的线程数，充分利用 CPU 资源，将消耗大量时间的业务交给自定义线程池完成，从而降低数据接入的时延。设计循环队列周期读写算法，使用读线程、写线程并发完成数据的周期读与写，实现动态增加队列大小，批量地缓存数据，解决了上游数据接入系统与数据分析系统速度不匹配问题，减小数据分析压力，提高数据分析的稳定性。周期接入数据便于规模化计算数据流和分析数据流的处理效率，本文提出一种数据动态配置策略，根据数据的实际处理能力与数据预期处理能力，计算出系统的周期数据处理效率，当周期数据处理效率大于 85%，系统负载过大，容易造成系统崩溃，需要加载节点，增加集群系统的处理稳定性，当周期数据处理效率小于 75%，系统负载过小，就要卸载节点来降低资源消耗。动态配置策略可以动态保证数据高处理能力，提高系统处理数据的稳定性，降低数据处理时延。

1　系统模型

1.1　系统架构

系统架构如图 1 所示，分为四大模块：数据接入模块、数据分析模块、数据控制模块、数据存储模块。数据接入模块主要是接入远程工业终端设备的生产数据，要求数据接入模块具有高并发、稳定处理的能力。数据接入模块主要使用 Netty 和 Kafka 技术，通过编写高性能 Netty 网络通信服务程序来接入数据，并且异步将数据推送到 Kafka 消息系统中。Kafka 消息系统是由多个 broker 组成的集群，有着更高的稳定性。数据分析模块由一系列分析子系统组成，以周期的模式来处理数据。数据控制模块控制数据的流转与配置，控制台下达用户的启动实例、增加配置的指令，配置中心存储和管理服务启动的配置信息，监控中心监控当前服务状态与当前周期的数据分析能力。数据存储模块由 Mysql、Redis 关系型和非关系型数据库组成，Redis 中存储热点数据，Mysql 中存储非热点数据。配置中心保存服务启动信息、接入系统连接信息、存储中心连接信息、周期数据配置信息。

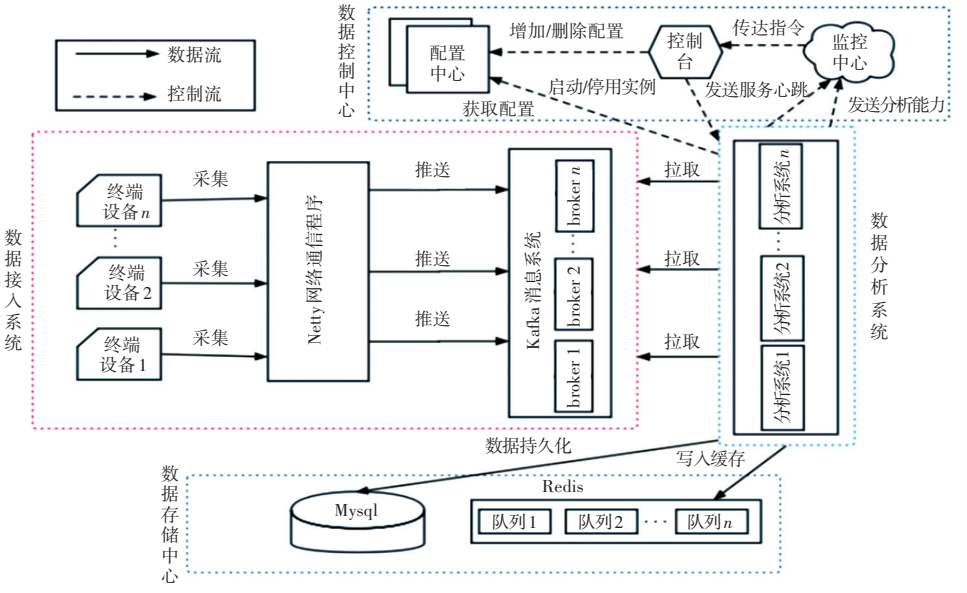

图 1　系统架构

Fig. 1　System architecture

当上游生产数据数量激增,控制台通过向配置中心写入服务启动配置信息,通过 Docker 客户端连接远程服务器,并且下达启动实例指令并虚拟化部署实例,实体启动之后,从配置中心获取服务启动信息,从而完成数据处理业务。若上游生产数据减少,控制台可以卸载服务节点,动态配置策略决定节点的数量。每个周期数据处理结束后,会向监控中心发送监控数据,包含服务节点信息、数据处理数量、周期等。数据接入服务在启动之初,会不断的向监控中心发送服务心跳,让监控中心感知到服务的存在。

1.2 数据接入性能优化

1.2.1 线程池优化

在数据采集的过程中用到大量的线程,会极大优化资源。如果直接使用创建线程的方式,不仅浪费时间,还会因为不明确业务处理时间,导致线程长时间阻塞,从而降低系统处理效率。数据预处理操作十分消耗线程资源与时间,使用 Java 自定义线程池根据业务来设置线程池参数,可以减少线程资源的开销。线程池的线程数量 TN_s,计算公式(1):

$$TN_s = TN_i \times (1 + \frac{W}{C}) \tag{1}$$

其中,TN_i 是 CPU 核心数;W 为线程等待时间;C 为线程所占用 CPU 时间。

数据接入系统的线程使用主要涉及到终端设备的连接、数据的预处理操作、数据推送操作,数据预处理耗时操作由自定义线程池来处理。NIO(Non-blocking I/O)是 Netty 的多路复用 IO 模型,NIO 业务线程与自定义线程池交互如图 2 所示。

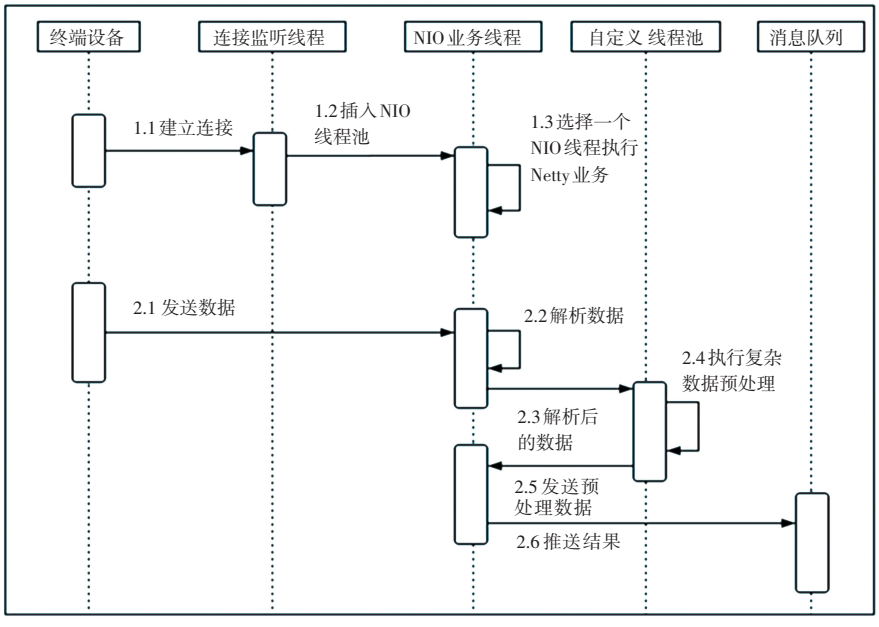

图 2 NIO 业务线程与自定义线程池交互

Fig. 2 NIO business thread interacts with custom thread pool

自定义线程池主要用来处理耗时的数据预处理业务,将处理好的数据交给 NIO 线程,由 NIO 业务线程将数据推送到下游消息队列。

1.2.2 通信协议设计

通信协议是数据接入的统一格式。简单、高效的通信协议会降低通信过程中的损耗,也可以使业务程序具有更好的扩展性。在传统的工业数据通信协议中,较多使用 MQTT 和 HTTP 协议,但 MQTT 协议在点对点的通信方式中存在一些劣势;HTTP 协议在通信的安全性方面有所缺失,无法确认通信方的身份。因此本文自定义协议用来保证高性能通信如图 3 所示。

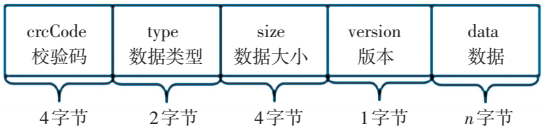

图 3 通信协议

Fig. 3 Communication protocol

(1)校验码(crcCode)

校验码主要是用来对连接进行校验,避免非本系统上游终端的连接操作,防止其他恶意连接获取资源。若校验码经常更换,上下游系统需要都进行修改,操作麻烦而且容易造成差错,故本文校验码采用静态固定值的方式,固定值为 0xabef0101。

(2)数据类型(type)

数据类型主要用来辨别传输数据的类型,由于工业数据的类型繁杂,不同类型的数据由不同的数据处理系统处理。数据类型一般分为文本数据类型和文件数据类型等。

(3)数据大小(size)

数据大小是为了解决 TCP 错误重传的问题。Netty 通信框架底层基于 TCP 实现通信,TCP 在通信过程中会因为网络等问题造成消息丢失,TCP 的重传机制会根据数据的大小进行重传。

(4)版本(version)

版本是系统迭代升级的标记,不同版本的系统解决的问题不一样,使用版本号可用来解释系统的功能性和适用性以及存在的缺点。

(5)数据(data)

数据就是终端设备采集的实时生产数据,其与数据类型有着强一致性。数据内容必须与数据类型一致,不然会造成系统的数据处理错误。

1.3　循环队列周期读写算法

上游海量数据流的接入速度远大于数据分析速度,将会造成系统的数据处理稳定性较差。为了解决这个问题及数据规模计算,本文设计一种循环队列周期读写算法。从 Kafka 指定 Topic 中获取数据流构建数据单元,放入循环队列;数据流写线程负责封装数据流单元,周期写入缓存队列;数据流读线程负责从缓存队列读数据。动态计算前两个周期队列的平均读写效率的比值,若比值过小,动态增加队列大小。队列中每个元素都是一个数据流对象,由数据流和数据流生产信息组成。循环队列结构如图 4 所示。

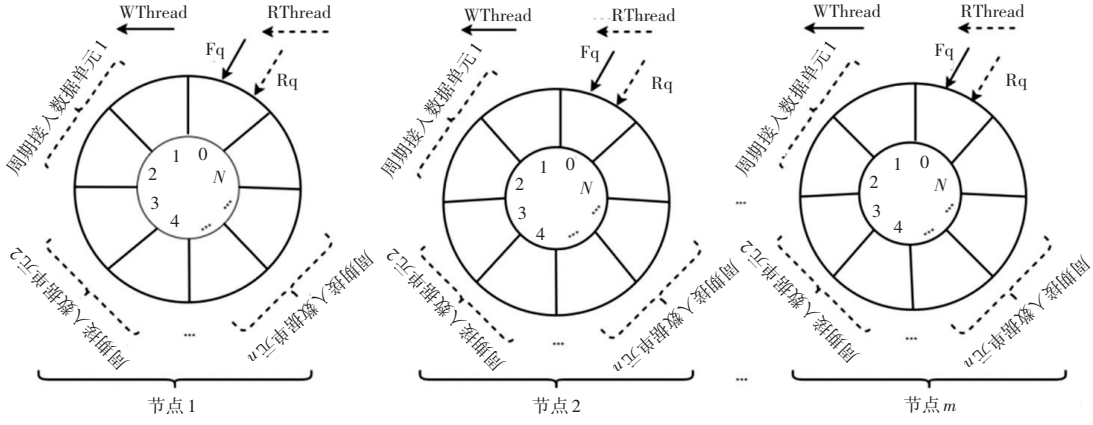

图 4　循环队列结构

Fig. 4　Circular queue structure

数据以周期的形式抓取,当周期抓取结束之后,写线程 WThread 批量将数据流组写入循环缓冲队列。读线程 RThread 一直从循环队列中读取数据。Fq 是头指针,Rq 是尾指针。为了减少队列加锁的粒度,对每个队列元素加入同步信号量解决并发问题。每个周期抓取的数据流都是一个数据流处理单元,使用循环队列读写算法,可以实现动态、批量的缓存数据,提高数据的接收与处理的效率。若存在 m 个节点,在 n 个周期内,假设 m 个节点同时启动,都经过 n 个周期的数据抓取,在第 i 周期, m 个节点抓取的数据数量集合为 $K_i = \{k_{i1}, k_{i2}, \cdots, k_{im}\}$, 队列写入时间(数据抓取到写入队列时间)集合为 $W_i = \{w_{i1}, w_{i2}, \cdots, w_{im}\}$, 队列读取时间(队列缓存到从队列取出的时间)集合为 $R_i = \{r_{i1}, r_{i2}, \cdots, r_{im}\}$,那么 m 个节点在第 i 个周期的平均数据写入速度为 WV_i, 式(2):

$$WV_i = \sum_{j=1}^{m} \frac{k_{ij}}{w_{ij}} \tag{2}$$

n 个周期节点的队列平均写入速度为 V_1, 式(3):

$$V_1 = \sum_{l=1}^{n} \frac{WV_l}{n} \tag{3}$$

m 个节点在第 i 个周期的平均读取速度为 RV_i, 式(4):

$$RV_i = \sum_{j=1}^{m} \frac{k_{ij}}{r_{ij}} \tag{4}$$

到 n 个周期节点的队列平均读取速度为 V_2, 式(5):

$$V_2 = \sum_{l=1}^{n} \frac{RV_l}{n} \tag{5}$$

计算 m 个节点在 n 个周期的队列平均读写速度之差 D, 式(6):

$$D = |V_1 - V_2| \tag{6}$$

当 D 无限趋近 0,则表示系统的数据处理稳定性更高;当 D 很大时,动态增加队列大小,缓解上游接入系统与数据分析系统的速度不匹配问题。

队列存储的数据流对象 d 包括数据流内容、数据流生成时间、周期标识、周期开始时间、周期结束时间、数据流生产信息等。

DS 为一个周期接入数据单元。抓取的数据流组转化成数据流对象组 DS = $\{d_1, d_2, \ldots, d_n\}$。

算法 1 循环队列周期写算法

输入 WThread 从 Kafka 集群指定 Topic 中获取的数据流构建成数据流对象组 DS

定义一个循环缓冲队列 Q ,数据流对象组

定义队列头指针 Fq、尾指针 Rq 和队列大小 N

定义开始时间 T_{start}、结束时间 T_{end}, 信号量 $P_{\text{Fq}} = 0, P_{\text{Rq}} = 0$

获取周期 T, 预计抓取数据数量 Num,Kafka 的 Topic 为 D_{Topic}

从 Kafka 指定 Topic 中获取数据

如果 DS.Size≥Num 则

遍历 DS

如果 DS 不为空 AND 队列未满 AND $T_{\text{end}} - T_{\text{start}} \geqslant T$ AND $P_{\text{Fq}} == 0$ 则

将数据流对象 DS 放入队列 Q.Fq%N 位置

将 Fq 指针移动到下一个空闲位置

$P_{\text{Fq}} = 1$

如果 $D_{i-2}/D_{i-1} \leqslant 50\%$ do // 根据公式(6),计算前两个周期 D_{i-2}, D_{i-1} 之比

增加队列 Q 长度为原来的队列长度的 1.5 倍

读线程 RThread 循环执行周期读算法,将一个周期接收的数据流读取出来进行数据周期处理。从队列中取出的数据流对象组 DE = $\{d_1', d_2', \cdots, d_n'\}$。

算法 2 循环队列周期读算法

输出 RThread 从循环队列中读取的数据流对象组 DE

获取周期数据标识 TFlag

定义队列获取数据 de

While Q 不为空 do

Q.Rq%N 赋值给 de

如果 de.flag==TFlag AND $P_{\text{Rq}} == 1$ 则 // 标识相同标识同一周期数据

将数据流对象 de 放进数据流对象组 DE

$P_{\text{Rq}} = 0$

移动 Rq 指针到下一个元素位置

如果 de.is_end is true 则 //如果该数据为周期结束标志,跳出循环

结束程序

Return DE

1.4 动态配置策略

数据分析模块是业务逻辑比较复杂的模块,十分消耗系统资源,影响整个数据的处理流程,是产生大量时延的过程。当数据量过大时,数据分析系统面对巨大的分析压力,导致时间延迟,甚至整个数据分析系统集群的崩溃。当数据量减少时,如果部署过多的数据分析节点,则会浪费过多的硬件资源。因此本文设计一种动态配置策略,解决数据分析系统资源过剩或资源短缺的状况。

设数据分析系统有 m 个节点,在一个周期 T 内, m 个节点数据的发送速率集合为 $V = \{v_1, v_2, \ldots, v_m\}$, m 个节点数据处理个数集合为 $N = \{n_1, n_2, \cdots, n_m\}$, 这 m 个节点预计处理数据个数集合为 $N' = \{n_1', n_2', \cdots, n_m'\}$, 且满足条件式(7):

$$\begin{cases} n_i \leqslant n_m' \\ \sum_{i=1}^{m} \frac{n_i}{\mathrm{T}} \leqslant \sum_{i=1}^{m} \frac{v_i}{m} \\ 1 \leqslant i \leqslant m \end{cases} \tag{7}$$

计算数据分析系统的周期数据处理效率为 E,公式(8):

$$E = \frac{\sum_{i=1}^{m} \frac{n_i}{n_i'}}{m} \times 100\% \tag{8}$$

其中, $\frac{n_i}{n_i'}$ 为第 i 个节点的负载率。

计算 n 个周期平均周期数据处理效率平均值 E_{av}, 式(9):

$$E_{\text{av}} = \frac{\sum_{i=1}^{n} E_i}{n} \tag{9}$$

根据 m 个节点在 n 个周期内的平均负载率正序排序,得到节点平均负载率排序集合为 $A = \{A_1, A_2, \ldots, A_m\}$。$E$ 越大,代表数据分析系统的数据处理性能越高,实验证明 E 为 75%~85%的时候数据处理效率以及硬件负载状况最好。当 E < 75%时,说明当前系统数据的处理效率比较低;当 E > 85%时,CPU 利用率过高,服务器资源可能会出现超负荷的状况,甚至会导致系统突然宕机。动态配置过程如图 5 所示。

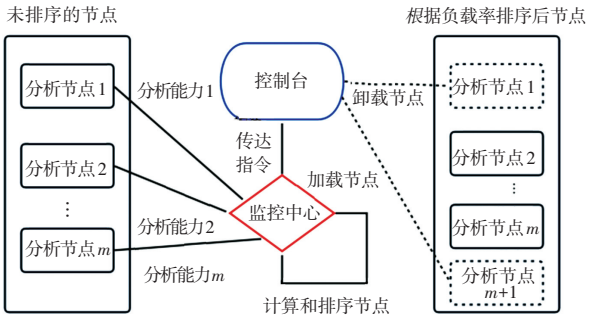

图 5　动态配置过程

Fig. 5　Dynamic configuration process

是否动态增加节点或减少节点由平均周期数据处理效率 E_{av} 决定。E_{av} <75%时，系统集群资源利用率过低，卸载平均负载率最低的节点；当 E_{av} >85%时，加载节点。

2　实验与分析

2.1　实验环境

为了验证系统的性能，以公交车实时摄像视频为数据基础，分别准备 20 台测试机，将测试的数据集分别等量的分发到 20 台机器上，并将 20 台机器分别部署同样的 Netty 客户端程序。启动 20 台测试机，同时向 Netty 服务端发送请求；Netty 服务端接收到数据后，对数据预处理，推送到 Kafka 消息系统，由数据分析系统群周期接入。数据分析系统初始只启动一个节点。硬件配置信息见表 1。

表 1　硬件配置信息

Table 1　Hardware configuration information

配置	详细信息
CPU	Intel Core i7-4700 CPU@ 3.60 GHz
内存	Kingston 12 GB 1 600 MHZ
硬盘	Scagatc 500 GB 7 200 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　服务配置信息

Table 2　Service configuration information

配置	详细信息
Kafka 集群服务器配置	采用三台机器为一个集群，端口 9092
数据分析系统信息	处理周期为：50 s，预计处理数量：100 张图片，数据类型为：公交车实时图片
数据缓存信息	一台服务器作为数据流缓存，另外一台服务器作为控制流缓存。

2.2　实验评价指标

一般使用每秒请求次数(TPS)来衡量评价系统的并发性能，本文使用 TPS 来评价数据接入系统的性能。T 为数据接入系统的每秒请求次数，并发数记为 CN，平均响应时间记为 t，计算公式(10)：

$$T = CN/t \tag{10}$$

T 的数值越大，代表系统的并发性能越高。

数据分析系统使用周期数据处理效率的标准差来反应系统的效率，考虑到 CPU 利用率过高，超过 90%，程序容易崩溃。实验证明，当最佳处理效率 E_{best} 为 80%时，数据处理性能较好。

系统的周期数据处理效率标准差 C 为计算公式(11)：

$$C = \frac{\sqrt{\sum_{i=1}^{n}(E_i - E_{best})^2}}{n} \tag{11}$$

C 的值越小，代表每个周期数据处理效率贴近最佳值，系统更加稳定。

数据处理速度也反应着系统的数据处理性能，设 m 个周期内 k 个数据分析系统节点分别处理的数据个数为 $N = \{n_1, n_2, \cdots, n_k\}$，分别使用的时间为 $T = \{t_1, t_2, \cdots, t_k\}$，忽略服务启动的时间损耗，那么数据处理速度为公式(12)：

$$V = \frac{\sum_{i=1}^{k} n_i}{\sum_{j=1}^{k} t_j} \tag{12}$$

2.3　实验分析

2.3.1　并发性测试分析

部署一台 Netty 服务器与 20 台客户端，每个客户端 1 024 个并发。分两种情况测试：第一种 1 次连接请求 1 000 次；第二种 1 次连接 1 次请求循环 1 000次，并发测试效果如图 6 和图 7 所示。

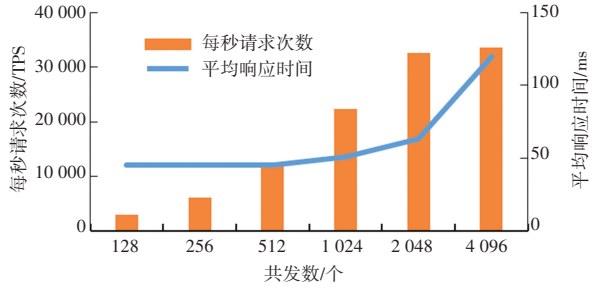

图 6　1 次连接 1 000 次请求并发测试

Fig. 6　Dynamic configuration 1 connection 1 000 requests concurrent test

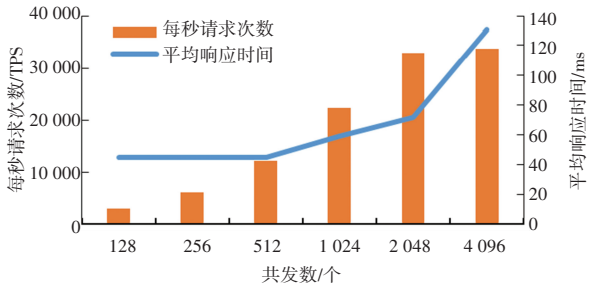

图 7 1 次连接 1 次请求循环 1 000 次并发测试

Fig. 7 1 connection 1 request cycle 1 000 concurrent tests

实验结果表明,1 次连接请求 1 000 次和 1 次连接 1 次请求循环 1 000 次,处理性能接近。2 万左右数据解析请求的时候,平均响应时间不到 200 ms,很小的采集时延给并发数据流的实时处理带来了极大的优势。

2.3.2 数据处理性能测试分析

上游以高速率、中速率、低速率 3 种速度模式交替推送数据。使用 1 个数据分析系统节点和 3 个数据分析系统节点做对比实验,循环队列周期读写算法 CRW 与未使用循环队列周期读写算法 CRW 的单个周期图片处理数量对比如图 8 和图 9 所示。

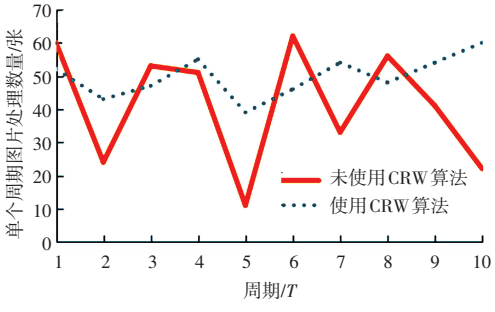

图 8 使用和未使用 CRW 算法的图片处理数量(1 个节点情况)

Fig. 8 The number of image processing with and without CRW algorithm in the case of 1 node

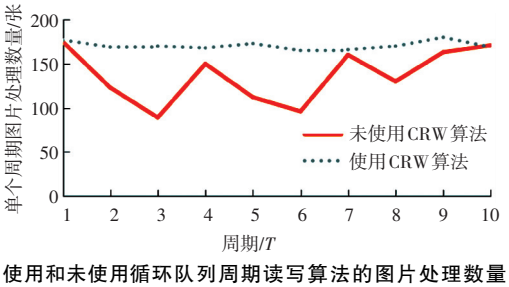

图 9 使用和未使用循环队列周期读写算法的图片处理数量(3 个节点情况)

Fig. 9 The number of image processing with and without CRW algorithm in the case of 3 node

由图 8 和图 9 可知,未使用 CRW 算法的 1 个节点平均每个周期的图片处理数量约为 63 张,3 个节点平均每个周期处理的图片数量约为 181 张。使用 CRW 算法的 1 个节点平均每个周期的图片处理数量约为 110 张,3 个节点的平均每个周期处理的图片数量约为 336 张。

和未使用 CRW 算法相比,当数据分析系统的节点数量越多,使用 CRW 算法数据处理的性能高且稳定。

随着周期增大一直给数据分析系统加载节点,未使用动态配置的图片处理速度如图 10 所示。

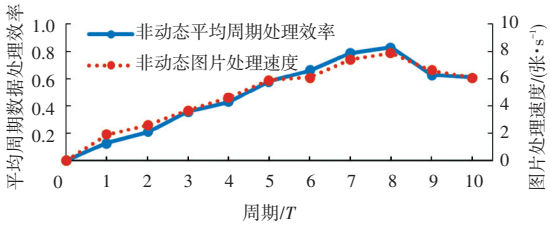

图 10 未使用动态配置的图片处理速度

Fig. 10 Image processing speed without using dynamic configuration strategy

经过动态配置策略节点调整之后的图片处理速度如图 11 所示。

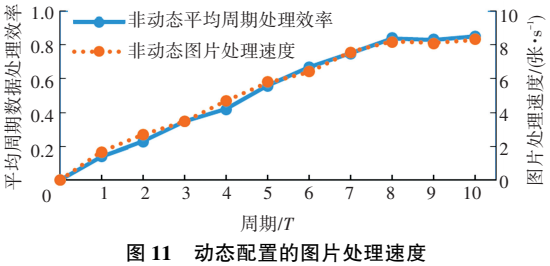

图 11 动态配置的图片处理速度

Fig. 11 Picture processing speed of dynamic configuration policy

由图 10 和图 11 可知,当使用动态配置策略算法调整服务节点之后,平均周期数据处理速率 E_{av} 为 80%以后,系统的图片处理速度达到稳定,故当 E_{av} 为 75%-85%之间是最为合理的,相比于未动态配置的情况,图片处理速度显著提高,CPU 利用率适中,并不会导致系统崩溃。

使用周期数据处理效率的标准差来表示系统的稳定性,如图 12 所示。可见,动态配置的周期数据效率标准差约为 1.9,非动态配置周期数据处理效率标准差约为 4.1,波动较大,动态配置之后的系统处理性能更加稳定,更加接近最佳值。

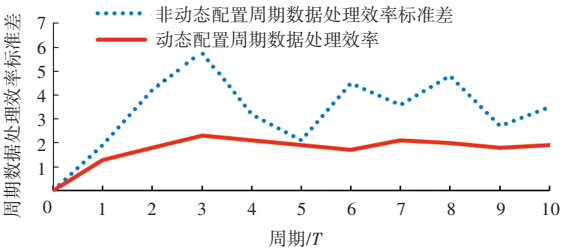

图 12 周期数据处理效率标准差

Fig. 12 Standard deviation of cycle data processing efficiency

3　结束语

本文提出一种数据接入与配置模型，从终端设备接收数据收集到服务器，通过自定义协议，减少传输损耗，降低传输时延。使用自定义线程池技术减少 Netty 自身线程处理业务的时间。使用周期批处理数据的方式来降低 I/O 消耗。使用循环队列周期读写算法提高数据处理的稳定性。设计数据分析系统的动态配置策略，动态加载或卸载数据分析系统的节点，从而影响周期数据处理效率。实验表明，当 E_{av} 为 75%～85%之间，图片的处理速度相比于未使用动态配置策略的情况，数据处理效率显著提高；周期数据处理效率标准差为 1.9 左右，每个周期的周期数据处理效率波动较小，系统的数据处理性能接近理想化，系统更加稳定。

参考文献

[1] 郭强，叶青，屈红冰，等. 基于 Netty 的智慧井盖监控系统设计与实现[J]. 工业控制计算机，2022，35(5)：33-34，38.
[2] 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.
[3] 任培花，苏铭. 基于 Kafka 和 Storm 的车辆套牌实时分析存储系统[J]. 计算机系统应用，2019，28(10)：74-79.
[4] 何倩，陈亦婷，董庆贺，等. 基于 MongoDB 的物联网接入云服务平台[J]. 桂林电子科技大学学报，2017，37(1)：1-7.
[5] 朱抗，何成昭，梁文超，等. 工业互联网高并发流式数据处理技术及应用[J]. 控制与信息技术，2021(5)：20-25.
[6] 崔炜. 一种面向云工作流的自配置算法[J]. 科技通报，2012，28(6)：25-27.
[7] 董正凯，叶彦斐. 基于循环队列缓冲的 Lora 通信数据终端设计与研发[J]. 工业控制计算机，2018，31(5)：18-19.