浙江理工大学学报，2020，43(2)：240-245
Journal of Zhejiang Sci-Tech University
DOI:10.3969/j.issn.1673－3851(n).2020.02.014

基于 Netty 和 Kafka 的 IOT 终端服务系统设计方案

张　娜[1]，史佳炳[1]，吴　彪[2]，包晓安[1]，文艺霏[1]
(1.浙江理工大学信息学院，杭州 310018；2.山口大学东亚研究科，日本山口 753-8514)

摘　要：解决 Netty 网络程序应用框架的 NIO 线程与业务处理线程间的耦合是提升服务器并发量的关键问题，为此提出了一种基于 Netty 和 Kafka 的 IOT 终端服务系统设计方案。该方案结合了 Kafka 消息中间件，使 Netty 只负责提供和管理 NIO 线程，其它的业务处理线程由 Kafka 的消费者端(Consumer)负责，并在 Consumer 中加入自定义业务处理线程池对复杂耗时业务逻辑进行处理，以解决因 Netty 线程被阻塞而导致终端请求堆积的问题。同时提出了一种自定义设备通信协议，在一定程度上提升了 Netty 编解码的速度，使 NIO 线程不会被阻塞。结果表明：与传统的 Netty 和 kafka 服务器系统方案相比，该系统方案对于上万级请求可以做到毫秒级响应，具有更快的协议编解码速度，并且在鲁棒性方面都有很好的表现。

关键词：Netty；Kafka；高并发；物联网；服务器
中图分类号：TP311.5　　**文献标志码**：A　　**文章编号**：1673-3851 (2020) 03-0240-06

Design scheme of IOT terminal server system based on Netty and Kafka

ZHANG Na[1]，*SHI Jiabing*[1]，*WU Biao*[2]，*BAO Xiaoan*[1]，*WEN Yifei*[1]
(1.School of Informatics Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China;
2.Department of East Asian Studies, Yamaguchi University, Yamaguchi 753-8514, Japan)

Abstract: Solving the coupling problem between the NIO thread and the business processing thread in Netty network application framework is the key to increasing the server concurrency. Therefore, a design scheme of IOT (Internet of things) terminal service system based on Netty and Kafka was proposed. This solution combines Kafka message-oriented middleware, so that Netty is only responsible for providing and managing NIO thread, while other business processing threads are under the charge of the Consumer of Kafka. And, a custom business processing thread pool was added in the Consumer to process the complex time-consuming business logic, so as to solve the problem of terminal request accumulation caused by blocked Netty thread. At the same time, a custom device communication protocol was proposed to improve the speed of Netty encoding and decoding to some extent, so that NIO thread will not be blocked. The results show that, compared with the traditional Netty and kafka server system schemes, this system scheme can respond to tens of thousands of requests in milliseconds, with faster protocol encoding and decoding speed, and has a good performance in robustness.

Key words: Netty; kafka; high concurrency; Internet of things; server

收稿日期：2019－09－29　　**网络出版日期**：2020－01－02
基金项目：浙江省重点研发计划项目(2020C03094)；浙江省自然科学基金青年基金项目(LQ20F050010)；浙江理工大学本科生科研创新计划重点项目(2019ZD-28)；浙江理工大学本科生科研创新计划一般项目(2019YB-24)
作者简介：张　娜(1977－　)，女，浙江奉化人，副教授，硕士，主要从事自适应软件、软件测试与智能信息处理方面的研究。
通信作者：包晓安，E-mail：baoxiaoan@zstu.edu.cn

0　引　言

随着物联网技术的迅速发展以及智能设备的普及，高吞吐量、低延时的高性能物联网服务器系统得到了广泛的关注[1]。当大量终端设备同时向服务器发送数据请求时，若不能快速处理客户端请求，将会导致大量请求的堆积，造成服务器的线程数量飙升，最终可能造成服务器宕机[2]。服务器系统的失效将会导致巨大的经济损失，良好的服务器架构设计可以有效提高服务器的并发性能[3]，因此设计高并发量的服务器系统具有重要的意义。

服务器架构设计的核心在于建立合适线程模型，利用好系统资源可以有效地提升系统的并发性能[4]，已有较多学者对此进行了研究。一方面，从对线程池的参数设置上进行调优已有不少研究，如：吉利等[5]选取服务器的 CPU 占用率和请求处理时间作为估量标准，对线程池中最佳线程数量进行估算，提出了一种线程池容量配置方案，保证系统在高负载和低负载的场景中，均能表现出较好的性能。另一方面，一些高性能网络通讯框架近年来也得到了国内外许多学者的关注，如：龚鹏等[6]基于异步网络通信框架 Netty，提出了一种快速构建高性能数据通信服务系统的设计方案，运用该方案设计的数据通信系统具有较高的可靠性、良好的移植性，以及高并发处理能力；金双喜等[7]通过引入一种基于 Kafka 消息队列的分布式消息处理机制，构建了一套具有高吞吐量并确实可靠的分布式消息系统；Zhang 等[8]通过使用 Netty 框架设计了一种可扩展为虚拟实验和物理实验的一体化系统；Ruben 等[9]在其实现的物联网分布式数据服务上，引入消息中间件来提升系统的性能和实时性。但是以上方法存在开发难度大、耗时业务逻辑阻塞 I/O 线程、消息容易堆积等问题。

本文针对以上不足，在 Netty 网络框架的基础上，引入 Kafka 消息中间件，以异步传送方式将 Netty 的 NIO 线程和业务处理线程进行解耦，避免了耗时业务逻辑阻塞 Netty 的 NIO 线程导致的消息堆积现象发生；在该架构的基础上，通过引入了一种快速解析的自定义设备通信协议和在 Kafka 的 Cousumer 模块中加入自定义线程池，进一步提高了物联网终端服务系统的并发处理能力。

1　线程模型和消息队列

1.1　Netty 的线程模型

在早期的 Java 语言中，使用多线程处理的主要方式是按需创建和启动新的线程来执行并发的任务单元，但这是一种在高负载下性能较差的原始线程模型。随后 JDK5 引入了 Executor API，它提供的线程池技术通过缓存和重建线程，可以减少线程创建和销毁带来的系统资源浪费，但并不能消除由线程上下文切换带来的开销。Java 原生的多路复用技术可以有效解决多线程中线程切换开销大的问题，但它存在“臭名昭著”的 epoll 空轮询 BUG，且 API 使用难度较大。Netty 网络框架底层封装了 Java NIO，具有使用简便、安全性高、性能优越等特点。并且 Netty 采用异步传输的方式，使用 Channel 作为传入或者传出数据的载体，使用 EventLoop 实现事件循环功能，每个 EventLoop 中都维护着一个 Selector 实例，采用 Reactor 单线程模型的工作模式，从而只用少量的 EventLoop 就可以支撑大量的 Channel，其中每个 Channel 在它的生命周期内只注册于一个 EventLoop，每个 EventLoop 都由一个线程支撑。EventLoopGroup 起到线程池作用，包含一个或者多个 EventLoop[10]。本文以包含 3 个 EventLoop 的 EventLoopGroup 为例对 Netty 线程模型的内部结构组成进行说明。示例线程内部模型如图 1 所示。在创建 EventLoopGroup 时就直接分配了 EventLoop，以及支撑它的初始线程，以确保它们在被需要时是可用的。并且 Netty 采用了串行化设计理念，在 Channel 创建时会自动分配专属的 ChanelPipeline 作为消息处理链的容器，其中消息的读取、编解码以及逻辑处理始终都由对应的一个 ChanelPipeline 负责，这就意味着整个流程不会进行线程的上下文切换[11]。所以提升系统性能的关键是不要阻塞这个线程，否则将会对系统整体的 I/O 处理造成负面影响。Netty 是个异步高性能的 NIO 网络框架，它并不是业务运行容器，所以本文采用消息中间件异步传送的方式将 NIO 线程与业务处理线程解耦。

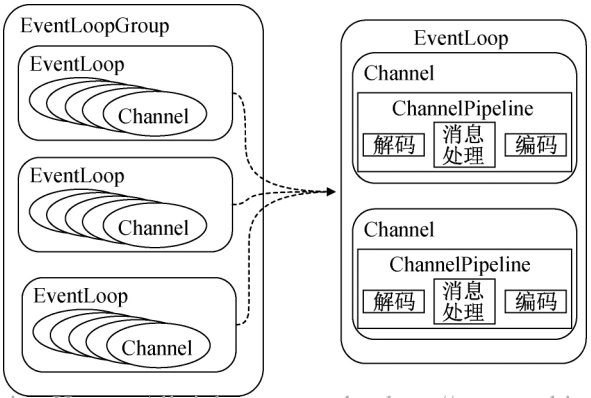

图 1　Netty 线程内部模型示意

1.2 Kafka消息队列

常用消息传递模式有队列模式和发布订阅模式[12]。在队列模式中,由多个Consumer从服务器中读取数据,但消息只会被其中一个Consumer所消费。在发布订阅模式中,消息会被广播给所有订阅了该主题(Topic)的Consumer。本文设计系统需要保证消息的顺序性,因此选用了Kafka的队列消息传递模式。相比传统消息队列,Kafka对于每一个Topic都至少保留一个用于缩放、并行化和容错性的分区,每个分区都只对应一个Consumer,所以Kafka可以同时提供顺序性保证和多个Consumer同时消费时的负载均衡,并且方便系统做分布式扩展。在一定的时间范围内,Kafka会维护所有由生产者(Producer)生成的消息,即使被消费了也不会从其所在分区中删除[13]。因此当终端设备的数据上传速率较消息处理速率快时,也能保证消息不丢失。另外,Kafka消息队列采用了Linux中的零拷贝技术(Zero-copy)提高了发送性能。传统的数据发送需要进行4次上下文切换,若采用Sendfile系统调用,数据可以直接在内核态进行交换,系统上下文切换将减少2次,可提升一倍的数据发送性能。本文引入Kafka消息队列,将对实时数据汇聚、大并发数据访问提供极高的吞吐量,且保证了数据的安全性。

2 系统设计

2.1 系统架构设计

本文设计服务器系统分为3个模块,分别是数据处理模块、消息中间件模块、业务处理模块。系统的整体架构设计如图2所示。

数据处理模块是基于Netty网络框架进行搭建的,充分利用了Netty编解码效率高和异步事件驱动的特性。当终端设备跟服务器通讯时,首先通过Netty的bossEventLoopGroup对终端设备的连接进行IP过滤,验证设备的合法性。接着由workerEventLoopGroup根据自定义通信协议对接收到的数据进行解码和解密操作,校验数据的有效性并封装数据成对象。同时对于每一条链路都添加等待超时处理逻辑,超时失效时间设为120 s,若服务器120 s没有接收到客户端发来的数据包,则立即关闭超时的连接减小系统的负载。

消息中间件模块引入了Kafka的消息队列,借助Kafka可以对数据流量削峰填谷的特性,完成了数据解析与数据处理的线程解耦。本文设计的消息

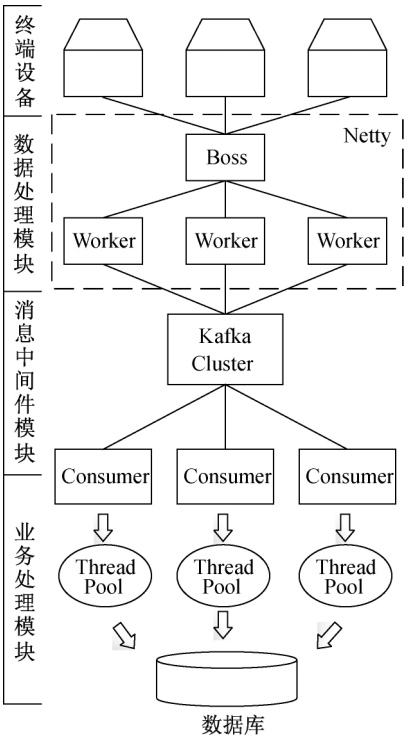

图2 系统架构示意

中间件模块是根据具体业务来为数据对象制定相应的Topic,Worker线程中调用Kafka的Producer把数据处理模块中生成的数据对象缓存到Kafka消息队列中。在业务处理模块中,Kafka的Consumer Group根据Topic拿到数据对象,并在Consumer中执行其具体的业务逻辑从而消费掉消息队列中的消息。但是在请求密集时间段,Producer会往消息队列中缓存大量的消息,若Consumer处理缓存消息的速度跟不上Producer生产消息的速度,将出现大量消息堆积的情况,因此本文服务器系统性能的另一瓶颈在于Consumer的消息处理能力。

业务处理模块主要涉及数据库的操作,等待IO处理往往会占用线程大量的时间。为了解决这个问题,本文系统在业务处理模块加入业务线程池,采用多线程的方式处理数据库IO操作,并设计了线程池调优的方案,有效减少了IO等待时间浪费的资源,提高了CPU利用率。

2.2 通讯协议设计

由于物联网终端一般都是低功耗、低内存、芯片级的设备,支持的网络协议也都比较底层,并且设备端的网络环境也是极其不稳定的,因此不适合使用HTTP等高级协议来传输数据[14]。本文采用TCP协议进行数据传输,TCP协议是流式服务,服务端收到的是一连串字节数据,会存在粘包拆包问题。本文将通过制定特殊的通讯协议来解决这一问题,

将消息分为消息头和消息体，并在消息头中存储消息长度。

自定义协议可根据业务需求进行设计，从灵活性的角度上看比其他应用层更加具有优势。同时在设计时要考虑后续系统升级时会出现的版本兼容性、扩展性等问题。一般消息分为消息头和消息体两个部分[15]。消息头为固定长度，用来描述消息属性信息，包括协议、消息长度、操作码、设备序列号以及校验码等信息。消息体为可变长度，是承载消息实体。本文设计了一个协议仅供参考。具体字段定义如表 1 所示。本文通过设计自定义通讯协议配合 Netty 半包解码器有效解决了 TCP 粘包和拆包的问题，使得传输码流更小，解析速度更快。

表 1　协议字段定义

字段名	字节数/byte	字段描述
version	1	协议版本
contentLength	2	消息长度
operationCode	1	操作码
serialNumber	4	设备序列号
content	自定义长度	消息体
CRC	1	校验码

2.3　业务线程池设计

业务处理逻辑中往往存在复杂耗时的数据库操作，由于 Kafka 的 Consumer 线程直接执行耗时的数据库操作，将会阻塞 Consumer 线程。若 Consumer 拿到的这些数据在规定时间内消费不完，则会被判为提交失败，数据就会回滚到 kafka 中，将会出现重复消费的情况。如此循环，数据就会越堆越多。本文方案设计了自定义业务线程池来处理耗时的业务逻辑，并通过线程数设置算法优化线程池，合理的分配了线程资源，用最合适的线程数处理队列中的消息，可以有效提高 Consumer 的消息处理速度。

Java 从 1.5 开始可以通过 Executor 框架来控制线程的启动、执行和关闭，极大地简化了并发编程的操作[16]。该框架中 Executors 类提供了 4 种可以用于创建线程池的静态工厂方法，分别是 newFixedThreadPool、newCachedThreadPool、newSingleThreadPoolExecutor 和 newScheduledThreadPool。其中，newFixedThreadPool 可以创建固定大小的线程池和可操控线程最大并发数，当线程池中的线程数达到其设定的最大数量时，新创建的线程会加入无界队列中等待，适合用于大负载的产品服务器。当线程池中某个线程因发生了异常而结束，线程池会立即补充一个线程继续执行任务。

设置线程池的最大线程数时需要避免“过大”和“过小”两种极端情况。若线程数设置过大，大量的线程将在相对不足的 CPU 和内存资源上竞争，这将会导致更高的内存使用量，而且还会耗尽 CPU 资源。若线程数设置过小，将导致大量处理器的空闲，无法执行工作而无法提高吞吐率。本文设计的系统包含具有 I/O 阻塞操作的任务，当 I/O 操作阻塞线程时，线程将停止执行直到 I/O 操作完成，因此线程等待时间与计算时间的比值是设置线程数大小的一个重要参考因素。可以通过以下方法来计算线程池大小：在某个基准负载下，分别设置线程数大小不同的线程池来运行应用程序，并记录对应的 CPU 的利用率[17]。线程池的最大线程数算法如式(1)：

$$N_{\text{threads}} = N_{\text{cpu}} \times U_{\text{cpu}} \times \left(1 + \frac{w}{c}\right) \tag{1}$$

其中：N_{threads} 表示线程的设置数，N_{cpu} 表示当前 CPU 核心数量，U_{cpu} 表示预期 CPU 核心使用率，w 任务等待的时间，c 表示计算时间。

2.4　服务端数据处理流程

终端设备与服务器通信的数据交互主要包括数据编解码、数据库操作、收发数据等。其中数据编解码、加解密等执行时间较短的操作交由 Netty 的数据处理模块执行，而耗时比较长的数据库操作则交由 Kafka 消息中间件模块转发到业务处理模块的自定义业务线程池中执行。服务器系统在启动时会监听一个端口。当终端设备与服务器建立连接后发送消息时，服务器将对消息进行处理，详细的消息处理流程如图 3 所示。其中：服务监听线程与 NIO 处理线程都是 Netty 框架中的异步非阻塞线程，分别负责消息的读写和根据通讯协议进行编解码；解码后的消息对象会交由 Kafka 消息中间件中等待处理；业务处理线程池存在于 Kafka 的 Consumer 中，对消息队列中的消息进行处理并返回结果。

3　实验结果和分析

3.1　实验环境

本文使用了一台八核高性能服务器搭载系统，服务器配置信息参数为：CPU 主频为 4.00 GHz，采用 Intel(R) Core(TM) i7-4790 K 处理器，32 G 内存，2 T 硬盘，操作系统版本为 Ubuntu 16.04.2。本文实验测试程序使用 Java 编写，JDK 版本为 1.8，Netty 使用版本为 4.1.17，Zookeeper 的版本为 3.4.12，Kafka 使用版本为 1.1.0。通过开源压力测试工具 Jmeter 模拟终端设备大量的 TCP 连

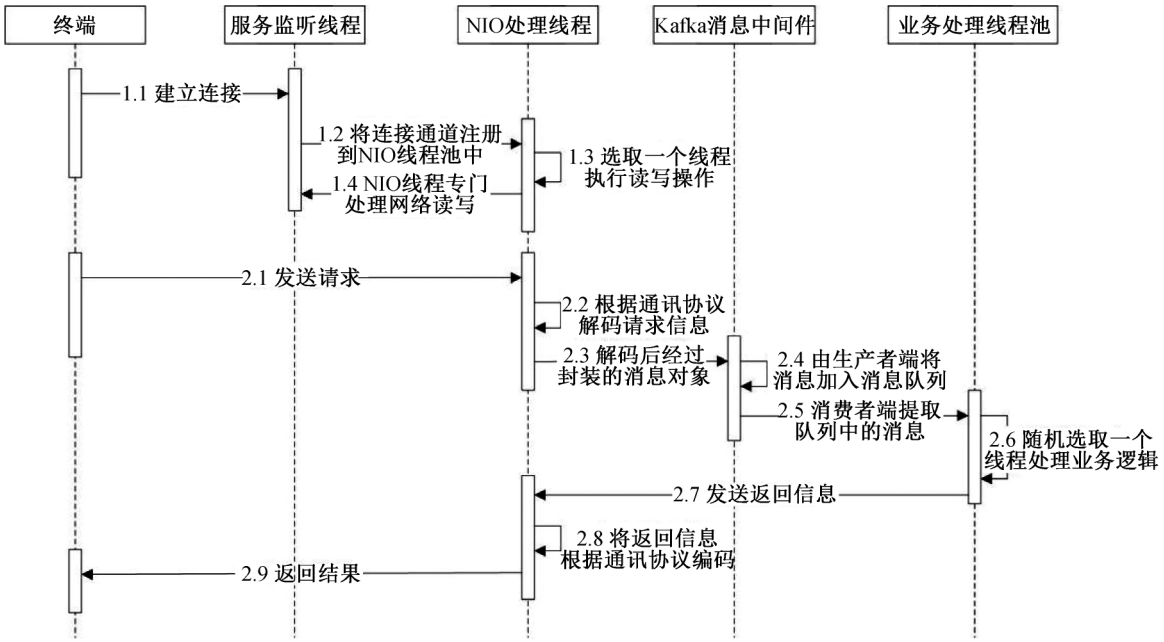

图 3 终端与服务器通讯数据交互流程图

接,使用多台压测机器在局域网中对系统进行性能测试。压测机器使用 3 台四核 CPU 的计算机,其配置信息参数为:Intel(R) Core(TM) i5-7200U 处理器,8 G 内存,256 G 固态硬盘,操作系统为 Win10。实验在局域网中进行,机器间通过百兆交换机相连,因此本文实验不考虑带宽的影响。

3.2 实验设计与分析

本文实验将基于 Netty、基于 Netty 和 Kafka 以及 Netty 结合优化后的 Kafka 三种方案进行比较。从系统平均响应时间和服务器 I/O 吞吐量两方面进行实验结果的比较分析。采用开源测试工具 Jmeter 模拟的终端设备每隔 1 s 发送 10 个数据包,三种方案的平均响应时间如图 4 所示,当并发量小于 2000 个时,采用 Netty 结合原生 Kafka 的方案并没有较大的优势,因为使用 Kafka 消息中间件会有一定的系统开销。当并发数量达到 3000 个时,采用 Netty 结合 Kafka 方案优势较为明显。基于原生 Netty 的实现方案因复杂耗时业务阻塞了 NIO 线程,导致响应时间过长[18]。采用 Netty 框架结合 kafka 的方案将复杂耗时的逻辑交由 Consumer 处理,极大的缩短了系统平均响应时间,但随着请求量的增大,在并发数达到 6000 个时,Netty 结合原生 Kafka 的方案由于 Consumer 处理消息速度不够快,出现了消息大量堆积的现象,使得响应时间急剧增加。本文方案在 Consumer 端加入业务线程池来对复杂耗时操作进行处理,Consumer 处理消息的速度提升比较明显,缩短了平均响应时间。

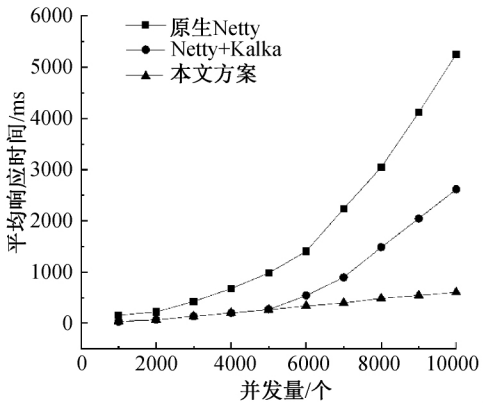

图 4 系统平均响应时间

在 IO 吞吐量方面,平均响应吞吐量如图 5 所示,当并发量达到 3000 个时,基于原生 Netty 的实现方案平均吞吐量达到最高值,但随着并发数请求数增大,服务器出现吞吐量下降的现象。当并发量 6000 个时,由于 Kafka 消息中间件出现消息堆积,基于 Netty 结合 Kafka 的方案平均吞吐量出现上升瓶颈。采用 Netty 框架结合优化后的 Kafka 方案有效解决了消息堆积的问题,在并发量达到 10000 个时依旧稳定运行。进行进一步极限压力测试,当并发量达到 16000 个以上才出现本文方案的性能瓶颈,主要是由于硬件性能的局限性造成的。

此外,当终端设备的连接数量达到 10000 个以上时,本文系统依旧可以稳定可靠地运行,而其他两种方案的响应时间急剧上升,并且出现超时异常,吞吐量也不再增加甚至出现下降的情况。这说明本文提出的 Netty 网络框架结合优化后的 Kafka 的方

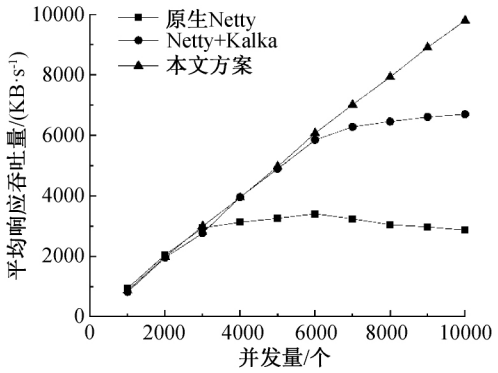

图 5　系统平均吞吐量

案，更加适用于物联网终端设备服务器系统面临的高并发业务场景。

4　结　论

本文提出了一种基于 Netty 框架结合 Kafka 消息中间件的物联网设备服务器系统设计方案，分析系统性能瓶颈及其产生原因，并采用自定义通信协议和 Consumer 自定义业务线程池设计了优化方案。Netty 和 Kafka 消息中间件保证了系统的稳定性和数据的安全性。自定义通信协议使得数据更加紧凑，解析速度更快。自定义业务线程池有效地提高了系统的高并发处理能力和吞吐量。

根据本方案设计的系统已投入智慧校园项目的使用，在实际场景的应用中未出现数据交互不稳定情况，且系统运行效果较好。目前的工作是服务器单机的优化方案，而本文使用的 Kafka 是一款高吞吐量的分布式发布订阅消息系统，今后的工作将围绕分布式集群展开深入的研究和实践。

参考文献：

[1] 包晓安，常浩浩，徐海，等.基于 LSTM 的智能家居机器学习系统预测模型研究[J].浙江理工大学学报(自然科学版)，2018，39(2)：224-231.

[2] 包晓安，魏雪，陈磊，等.基于 mean-variance 的服务集群负载均衡方法[J].电信科学，2017，33(1)：1-8.

[3] 包晓安，胡星，张娜，等.基于控制系数的交通信号动态配时研究[J].浙江理工大学学报(自然科学版)，2014，31(9)：517-520.

[4] Zhang Y，Yu L J，Li Y Q，et al. Optimization design method of communication service system for vehicle remote monitoring based on Netty pattern[C]//2017 Chinese Automation Congress (CAC)，2017. New York：IEEE，2017：682-686.

[5] 吉利，潘林云，刘姚.线程池技术在网络服务器中的应用[J].计算机技术与发展，2017，27(8)：149-151.

[6] 龚鹏，曾兴斌.基于 Netty 框架的数据通讯服务系统的设计[J].无线通信技术，2016，25(1)：46-49.

[7] 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.

[8] Zhang S，Zhu S. Server structure based on netty framework for internet-based laboratory[C]// 2013 10th IEEE International Conference on Control and Automation (ICCA). Hangzhou：IEEE，2013：538-541.

[9] Ruben C H，Rafael D S J，Maristela D H，et al. Distributed data service for data management in internet of things middleware[J/OL]. Sensors，2017，17(5)：997. https://doi.org/10.3390/s17050977.

[10] Maurer N，Wolfthal M A. Netty in Action[M]. Greenwich，USA：Manning Publications Co，2015：70-98.

[11] 李林锋. Netty 进阶之路[M]. 北京：电子工业出版社，2019：147-153.

[12] 王岩，王纯.一种基于 Kafka 的可靠的 Consumer 的设计方案[J].软件，2016，37(1)：61-66.

[13] 甄凯成，黄河，宋良图. 基于 Netty 和 Kafka 的物联网数据接入系统[J/OL].计算机工程与应用. http://kns.cnki.net/kcms/detail/11.2127.TP.20190319.1611.014.html.

[14] 包晓安，曹云棣，张娜，等.基于格分布方差的多目标云工作流调度算法[J].电信科学，2019，35(2)：1-13.

[15] Duan D H，Huang M X，Mu Y Z. Research for building high performance communication service based on netty protocol in smart health[C]//Proceedings of the 9th International Conference on Signal Processing Systems：ICSPS 2017. New York：ACM Press，2017：18-21.

[16] 黄天天，刘波.基于 Netty 框架的农村应急广播高并发数据处理[J].湖南农业科学，2017(9)：100-104.

[17] Goetz B，Peierls T，Bloch J，et al. Java Concurrency in Practice[M]. New Jersey：Addison-Wesley，2006：93-149.

[18] 顾振德，刘子辰，龙隆，等.基于 Netty 的 IoT 终端通信服务系统设计[J].计算机应用与软件，2019，36(4)：135-139.

(责任编辑：康　锋)