文章编号：2095-2163(2024)02-0062-07 中图分类号：TP391 文献标志码：A

面向流式数据的弹性缓存管理系统设计

陶雨棚[1]，陈庆奎[1]，黄 陈[2]
（1 上海理工大学 光电信息与计算机工程学院，上海 200093；2 上海市第一人民医院，上海 201620）

摘 要：腹腔手术过程中需要维持腹腔的恒压，在利用物联网技术和云计算精准、实时地辅助医生完成恒压的过程中，存在大量流式数据，一些高负荷场景中存在缓存数据积压问题，导致的系统延迟将影响医疗器械的稳定性。在腹腔镜手术恒压器械中，缓存发挥着至关重要的作用，可以提高系统的吞吐量和响应速度，并减少对下游系统的压力。在实际应用中，上游传感器端由于网络延迟、数据内容等因素导致该单元的数据产出往往是不稳定的，在这种场景下常用的静态缓存策略易造成数据积压或缓存浪费，进而影响资源利用率。为此，本文提出以协调集群中上、下游节点间的资源为基础，通过数据积压预测结合协同均衡的策略，下游单元能感知上游单元数据产出后的负载变化趋势，预先调整资源分配；上游能通过管理单元的均衡调整，及时调整目标缓存节点及数据处理率。利用弹性缓存管理后，缓存占用比不随上游传感器数据量的变化而变化，数据得到了及时处理。

关键词：缓存管理；弹性；数据积压问题；流式数据

Adaptive management for backpressure in data stream caching

TAO Yupeng[1], CHEN Qingkui[1], HUANG Chen[2]
(1 School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China; 2 Shanghai General Hospital, Shanghai 201620, China)

Abstract: In the process of abdominal surgery, the maintenance of constant pressure in the abdominal cavity requires a lot of sensor information to be accurately and real-time assisted by the doctor using IoT and cloud computing technology. In some high-load scenarios, there is a problem of cache data accumulation, which leads to system delays and affects the stability of medical devices. In this abdominal laparoscopic constant pressure device, the cache plays a crucial role. It can improve the system's throughput and response speed and reduce pressure on the downstream system. In actual use, the upstream sensor end often experiences unstable data output due to network delays, data content, and other factors. In this scenario, common static cache strategies are prone to data accumulation or cache waste, affecting resource utilization. Therefore, this paper proposes a strategy of data accumulation prediction combined with cooperative balance based on coordinating resources between upstream and downstream nodes in the cluster. In this management plan, the downstream unit can perceive the load change trend after the data output from the upstream unit and adjust resources beforehand, and the upstream unit can adjust the target cache node and data processing rate in a timely manner through the balance adjustment of the management unit. After using elastic cache management, the cache occupation ratio does not change with the upstream sensor data, and the data is processed in a timely manner.

Key words: cache management; adaptive; backpressure; data stream;

0 引 言

物联网技术允许各种传感器和设备之间的互联互通，并通过互联网将这些数据传输到远程服务器，产生流式数据[1]。医疗物联网的应用场景下需要实时的云端反馈以辅助医生的诊疗过程。一个完善的恒压手术流程中，上游需要不断获取各个传感器的信息，包括监控当前的腹腔气压值、当前阀门的开闭限度、当前腹腔内二氧化碳的含量等等；下游云计算单元首先需要通过机械学习完成对患者基础信息

基金项目：国家自然科学基金(61572325)；上海重点科技攻关项目(19DZ1208903)；上海交通大学医学院医工交叉项目。

作者简介：陶雨棚(1998-)，男，硕士研究生，主要研究方向：医工交叉、分布式系统、人工智能；黄 陈(1978-)，男，博士，教授，博士生导师，主要研究方向：普通外科基础、临床研究。

通讯作者：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、并行数据库、并行理论等。Email：chenqingkui@usst.edu.cn

收稿日期：2023-02-14

聚类，再通过 DTW 时序模型实时地完成手术过程中腹腔压预测，最终通过控制理论 PID 计算出合适的阀门等级，完成恒压过程[2]。

流式计算中会对传感器、患者基础信息进行缓存以提升实时性，而缓存管理是提升吞吐率的难点之一，不合理的缓存管理轻则导致数据丢失，重则导致节点崩溃。相对于静态的管理方式，弹性管理方式更适合流式计算系统，根据上下游负载关系动态的管理缓存，避免缓存积压和浪费，提升系统的灵活性和可靠性。数据流具有实时性、高速性、多样性、并发性和不确定性等特征，这些特征对数据处理和系统设计都提出了较高的要求。在并发数据流处理领域中，需要通过合理的系统设计和算法优化，提高系统的处理效率和稳定性。为解决此流式数据积压问题，本文提出具有以协调集群中上、下游节点间的资源为基础，通过数据积压预测结合协同均衡的策略，优化数据处理率和资源利用率，设计了面向流式数据负载的弹性缓存管理方法。

1　相关研究

相关理论研究主要有利用各个计算资源之间的关系建立资源图，针对调整过程中调节力度和调节时机，通过分析输入负载变化趋势和反馈机制进行调和[3]。数据反压的实际应用中，孙一佳等[4]提出基于数据迁移策略的反压方法；Hanif 等[5]结合分布式计算引擎的反压方法；王绪亮等[6]通过控制数据源产出率缓解数据积压的应用。在数据层面使用合适的缓存替换方法亦可降低数据积压，在命名数据网络中通过多个维度计算缓存价值，以兴趣包的形式作为所有节点的参考因素，最终选择数据包缓存的节点[7]。从计算资源考虑，通过支持系统信息收集策略和处理策略的定制化，调度合理的计算资源处理积压节点，亦可解决数据积压问题[8]。

在深度学习领域中，较为完善的组件 Alluxio 旨在加速大规模分布式存储系统的访问，以提升生产率和成本优化[9]。在解决缓存数据积压时，主要有利用分布式计算生成存储系统中数据块的关联规则集[10]，董文菁等[11]还通过改善其随机访问的能力。这些解决方案停留在缓存数据层面，并没有真正考虑到上下游负载关系。在流计算引擎中，数据积压问题也不可避免。虽然此类计算引擎具有庞大且高性能的计算集群，但当数据流突变性较大时会反复进行反压，从而导致数据震荡，熊安萍等[12]提出可变队列的反压机制，主要参考 TCP（Transmission Control Protocol）拥塞控制的 AMID（Additive-Increase/Multiplicative-Decrease）规则。利用计算引擎特有的血缘机制提出动态逐级反压，无状态拓扑数据重复，自适应拓扑替换，延迟持久化队列等基于 Storm 的解决方案[13]。这些解决方案必须在一个分布式计算引擎框架中实现，对小型应用不是很友好且存在资源浪费的可能，但其反压机制值得借鉴。

相关工作中各个成果都具有一定针对性，如针对深度学习、命名缓存空间、大型计算引擎、控制特定组件等，缺乏对负载的把握，导致部分缓存节点出现数据积压。本文研究具备上述特征的弹性缓存管理方法，具有通用性，适应当前的缓存特征并结合各类资源。

2　弹性缓存管理系统

弹性缓存管理系统的弹性主要体现在：

（1）缓存资源、上下游组件皆是插拔式的，可根据应用需求添加、移除，避免资源浪费；

（2）内部均衡方案是可配置、可扩展、可定制的，在不同缓存场景下运用合理的均衡方案以提升反压效果；

（3）在缓存层面，数据上下游在反压过程中也是弹性的，在反压过程中随着系统负载动态分布数据负载。

本文设计了一个具有预知数据积压并做出均衡调整的弹性管理系统。首先，拥有全局感知能力，从其他组件获取上下游的负载状态的同时，能汇总当前缓存资源的状态；计算均衡方案依赖计算资源，需要一个高效计算负载的数学模型，以节省计算资源，不仅可以加快系统均衡决策过程，还能避免重复计算庞大的数据；系统能依赖各类信息作出均衡方案，通过协调上下游的方式执行均衡方案。

因此，弹性缓存管理系统主要包括三大模块：

（1）用于缓存数据写入以及配合反压协调器调整已存储数据的核心管理模块；

（2）用于综合分析各类监控数据，结合时序模型计算均衡方案的时序预测模型；

（3）用于接收管理单元的触发信息，指挥上下游均衡调整并反馈结果至管理单元的反压协调器模块。弹性缓存管理系统的总设计如图 1 所示。

2.1　反数据积压协调器模块

协调器协助管理器分发上下游调整信号给各个组件，针对各个组件提供相应的调整信号。写入缓存的上层组件无法通过调整自身写入速率完成该调

整,则使用延迟写入,一定程度上降低数据积压的可能性。这一类调整方案都由管理器综合计算得出,而各个组件支持的调整方法会在系统初始化时注册给管理器。此外,数据反压协调器还可以使用人工干预的方式将调整信息分发给上下游。例如,可以连接报警系统,在系统出现异常时向运维人员发出提醒,并由运维人员手动调整系统设置以解决问题。

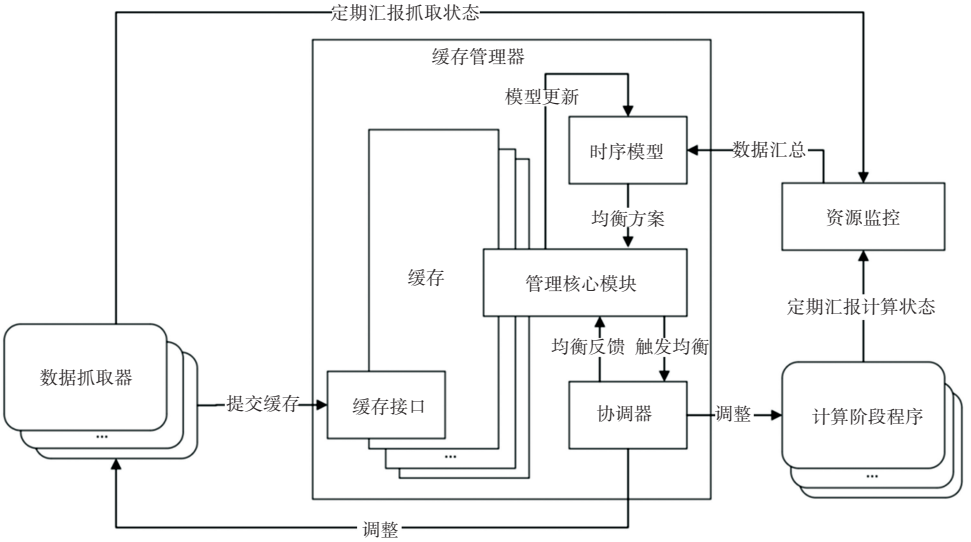

图 1 弹性缓存管理系统设计

Fig. 1 High level design for adaptive management system

发出的调整信号包括:

(1)调整类型:即对组件的调整操作,如提高写入速率、降低读取速率、暂停/恢复操作等;

(2)调整目标:指定调整的组件;

(3)调整值:即对组件执行的调整操作,如降低读取速率的百分比、写入速率的上限等;

(4)执行时间:指定该调整的生效时间;

(5)执行周期:指定该调整的持续时间;

(6)调整说明:对调整目的、结果等的说明,方便后续追踪和管理。

2.2 缓存管理模块

核心管理模块在该系统中负责资源管理的分配,在时序模型预测后发现可能的不均衡,及时通过自身掌握的资源信息进一步明确均衡方案,并将该方案递交到协调器,协调器完成整个调整过程后记录协调结果。若发生均衡失败,则通过相关日志分析其原因后尝试方案修整并再次触发协调器。

缓存系统的基本资源管理,包括缓存数据的增删改查,缓存的过期管理,缓存数据状态监控,缓存数据的自动加载,缓存数据的清理与回收。这些基础功能可以通过运用成熟的 Redis Cluster 或者 MemCache 等缓存供应商提供的接口实现。弹性缓存管理系统的核心针对数据积压时,在缓存供应商的基础上实现相关逻辑,完成缓存管理和协调。缓存管理内部结构如图 2 所示。

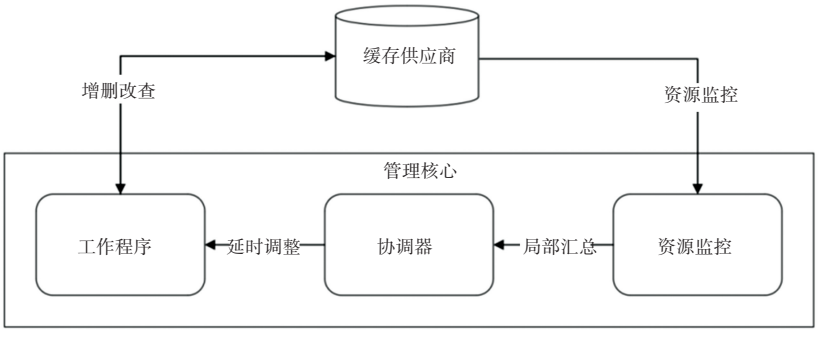

图 2 缓存管理核心内部结构

Fig. 2 Internal structure of cache management core

工作程序组件的主要功能是与缓存供应商交互完成缓存的增删改查，还实现了下游控制上游的协调方式。工作程序可以根据协调器最终确定的协调方案，设计一个响应延时以达到限制上游生产速率，此种控制方式取决于吞吐率与资源利用率之间的协调。若上游速度过快，工作程序节点另有实现 WAL（Write-Ahead Log）的能力，在一定阀值之内先将当前的缓存写入持久化容器，以防止数据丢失[14]。

资源监控器组件主要负责从缓存供应商定期读取相关资源信息并进行局部聚合，为均衡器提供可靠信息。均衡器组件是管理器的核心，负责协调不同模块之间的信息流动，并结合时间序列模型预测信息、缓存资源状态，制定最终的上下调整策略，并将其分发给数据反压协调器。具体的，管理模块可以使用数据挖掘算法分析时间序列模型预测信息、缓存资源状态以及上下游状态，从中提取有意义的信息并构建模型。在具体的调节方式上，文献[11]侧重解决了调整延迟和过度调整导致颠簸的问题，通过数据关联并利用图计算对多项数据进行数据挖掘，综合给出调整方案。弹性缓存管理系统在其基础上通过上下游数据流的时间序列分析，加速数据挖掘和资源图运算过程，进而解决综合处理速率提升模型适应性的问题。

数据反压均衡方案主要通过以下方式调整上下游组件：

（1）通过调整带宽限制来控制数据流的流量，从而避免过大的流量压力；

（2）通过定期主动清理缓存来释放系统资源，从而避免过多的数据堆积；

（3）根据当前的系统状态启用或禁用上下游组件，以缓解流量压力；

（4）根据当前的系统负载情况动态调整上下游组件的资源配置，以提升系统性能；

（5）应用削峰策略，通过延迟或拒绝一部分流量来缓解系统压力。

以上 5 个默认方案能完成数据反压的操作。弹性缓存管理系统的通用性使得反压方案可以订制，用户可根据当前应用场景开发适用性更强的方案，亦可使用优先级方案，组合或层级调用各个反压方案，这也是弹性缓存管理系统体现的弹性所在之一。

均衡器联系各方数据并进行均衡处理的分发器工作流程如图 3 所示，均衡处理是一个持续不断的过程，只有当出现均衡失败时需要管理员介入，因为缓存系统极有可能出现内存溢出，导致数据丢失等等问题。

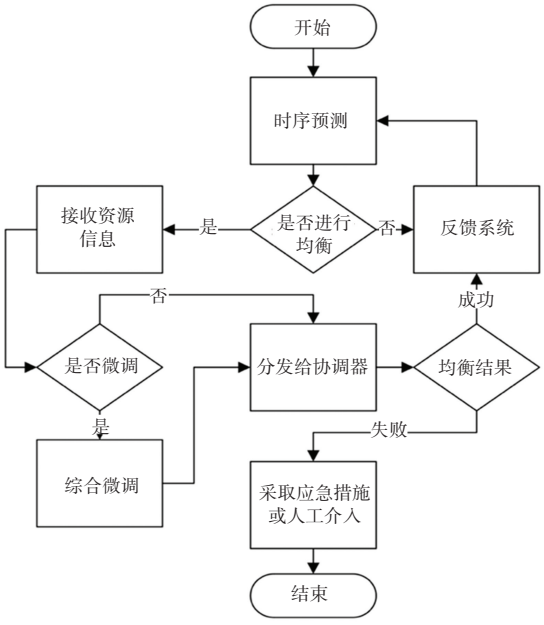

图 3　分发器工作流程

Fig. 3　Workflow of resolver

2.3　负载感知单元

结合系统内其他模块，负载感知单元的实现流程：

（1）收集节点的负载信息，包括当前周期内数据流的负载信息、CPU 利用率、内存占用状态等；

（2）根据收集到的信息，计算出当前节点的负载情况，并生成负载报告，包括节点的负载级别、主要指标信息；

（3）将负载报告发送给其他节点或管理中心，并接收其他节点或管理中心发送的负载信息，节点之间就可以相互了解彼此的负载情况，从而协助上下游的负载感知和协同控制。

2.3.1　流式数据模型

大规模的终端设备具有并发周期的特征，因此集群对 AI（Artificial Intelligence）并发数据流按照周期进行划分，产生的数据包内部形式如图 4 所示。数据流周期抓取的单个数据之间形式并不统一，数据主要以元数据，元数据信息的形式，具体数据流模型为

$$X = \{(x_1, s_1), (x_2, s_2), \cdots, (x_n, s_n)\}, n \in N \quad (1)$$

其中，(x_i, s_i) 表示第 i 条数据流以及其对应的元数据和元数据信息。

可将数据流的元信息封装在数据包的头部，元数据则以元数据，元数据信息的形式作为数据包的承载内容。

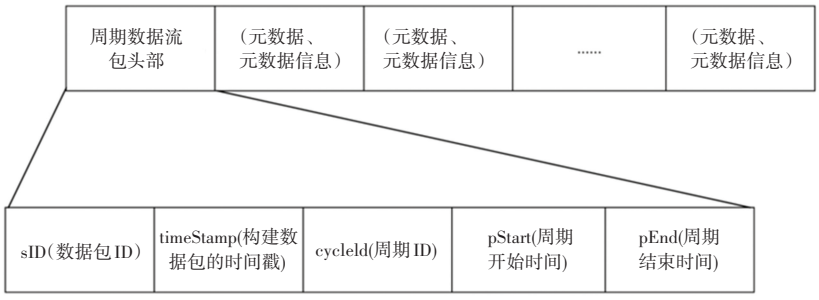

图 4 数据包内部形式

Fig. 4 Format of data package

2.3.2 负载感知模型

从监控系统中获取的监控字段为动态 CPU 占用率，动态内存占用状态，静态 CPU 峰值，静态内存最大容量，动态平均网络带宽，静态网卡带宽，动态节点处理速度。针对不同的应用每个指标对负载的影响权重都不相同。通过数学模型描述负载报告，负载报告综合当前周期内数据流的负载信息、CPU 利用率、内存占用状态。节点负载信息模型从监控单元获取当前周期节点的 CPU 利用率为 u_t，内存占用率为 m_t，使用式(2)来计算当前节点的负载值。

$$L_n = w_1 u_t + w_2 m_t \tag{2}$$

其中，w_1、w_2 为权重参数，可以根据实际情况进行调整。

假设数据流 X 包含 n 个数据条目，每个数据条目的大小为 s_i，数据流在当前节点的平均处理速度为 v_t，T_x 表示周期数据流在当前节点平均需要的处理时间，式(3)：

$$T_x = \sum_{i=1}^{n} \frac{s_i}{v_t} \tag{3}$$

数据传输时间为数据打包以及数据写入到缓存节点的总时间，缓存的写入写出相对传输可以忽略不计。数据包的周期信息可以从数据包的头部获取，分别为周期开始时间和结束时间，转换为数学变量的形式 p_s、p_e；另外网络带宽信息也可从监控系统中获取，由于系统波动性网络带宽信息为周期内的用于数据传输的平均网络带宽，转换为数学变量为 B_t，则平均数据传输的时间 T_{trans} 可以用式(4)表示：

$$T_{\text{trans}} = p_e - p_s + \sum_{i=1}^{n} \frac{s_i}{B_t} \tag{4}$$

数据流的负载度 L_x 可定义为基于最大容许延迟时间(P_{base})的每条数据处理和上游数据传输所产生延迟的平均值，式(5)。当 $T_x - T_{\text{trans}} \geqslant 0$ 时，其负载度为 L_x 的计算量；当该值小于 0 时，此时处理速度快于传输速度，不会造成数据积压，因此 L_x 为 0。

$$L_x = \begin{cases} \dfrac{\sum_{i=1}^{n} \dfrac{s_i}{v_t} - \left(p_e - p_s + \sum_{i=1}^{n} \dfrac{s_i}{B_t}\right)}{n P_{\text{base}}}, & T_x - T_{\text{trans}} \geqslant 0 \\ 0, & T_x - T_{\text{trans}} < 0 \end{cases} \tag{5}$$

综合数据流的负载度 L_x 和节点的负载度 L_n，则该节点在时刻 t 的综合负载度为 L_t，式(6)：

$$L_t = \begin{cases} w_1 u_t + w_2 m_t + \dfrac{\sum_{i=1}^{n} \dfrac{s_i}{v_t} - \left(p_e - p_s + \sum_{i=1}^{n} \dfrac{s_i}{B_t}\right)}{n P_{\text{base}}}, & T_x - T_{\text{trans}} \geqslant 0 \\ w_1 u_t + w_2 m_t, & T_x - T_{\text{trans}} < 0 \end{cases} \tag{6}$$

在并发情况下，假设有 h 个数据流，L_{ti} 为在 t 时刻第 i 个数据流所在的负载度，并发数据流在同一服务节点内并发线程共享 CPU 利用率和内存占用率，则在时刻 t 的负载度 L_{pt} 可表示为式(7)：

$$L_{pt} = w_1 u_t + w_2 m_t + \sum_{i=1}^{h} L_{ti} \tag{7}$$

将计算得出的负载报告值按照自定义区间划分负载等级，结合默认数据反压策略以及自定义反压策略的调节力度，匹配不同的负载度。负载感知单元负责提供负载感知能力，量化的负载度为上下游的负载匹配提供数据依据，具体的积压度量化表可以根据结果划分，见表 1。

表 1 积压度量化表

Table 1 Load level

负载度范围	负载度描述
[0~0.3)	0
[0.3~0.6)	1
[0.6~0.9)	2
[0.9~1)	3
[1~+∞)	4

负载感知单元可以将负载报告通过消息队列发布在负载报告主题中，以便其他节点或管理中心订

阅该主题并及时了解节点的负载情况。负载报告的数据形式为负载度，时间戳，内存状态，订阅者通过校对当前时间与时间戳确认负载度的时效性，并通过管理中心实现数据反压策略。轻量级的负载报告利用系统外的消息队列提供积压度数据，减轻系统的处理需求的同时提供实时的负载报告。

3　实　验

3.1　实验环境与配置

本文通过.NET 和 Python 实现弹性管理系统和时序预测模型。时序预测模型采取自动平滑模型，时间粒度选择为分钟。数据来源为项目开发中的实时流式数据，单个数据包流量较大，具有以天为循环的季节性等数据特性，具有典型意义。测试服务器采用 4 核 8G 的 Azure 云服务器，操作系统为 centOS7，.NET 版本为 6。该应用为 CPU 型且需要一定的内存完成计算，负载感知单元所需的服务器静态信息见表 2。

表 2　服务器静态信息

Table 2　Static information of the server

静态信息	初始值	静态信息字段描述
sCPU	4 * 2.3 GHz	静态 CPU 峰值
sBand	100 MB	静态网卡信息
sMem	8 G	静态内存最大容量

3.2　实验结果分析

3.2.1　性能指标

流式数据缓存的积压度与上游写入速度和下游处理速度息息相关，当写入量过大，远远超过下游处理速度的时候，用于缓冲的缓存极易产生缓存溢出的问题。应用本文设计的弹性缓存管理系统后，直观上随着数据流的变化缓存突增的幅度会降低很多，通过量化缓存曲线的变化率可验证弹性缓存管理系统的可用性。记 t 时刻的缓存量为 Mem_t，$\triangle t$ 为时间粒度，则该曲线的平滑度 S_t 可利用离散化的平滑值求得，式(8)：

$$S_t = \sum_{t=2}^{n-2} \left[(Mem_{t+2} - 2Mem_{t+1} + Mem_t) + \triangle t \right]^2 \quad (8)$$

3.2.2　结果分析

负载感知单元通过负载模型计算负载度，此实验应用为 CPU 型，CPU 利用率 w_1，内存占用量 w_2 分别设置为 0.65，0.8，容许的延迟时间 P_{base} 设置为 200 ms，即尽可能保证此应用的 CPU 运算的吞吐量，并防止内存出现溢出导致系统崩溃。CPU 在进行高负荷运算时，内存占用率随之增加，负载感知单元提供的负载度表明此时的负载度波动明显，当负载度达到 0.6 及以上时，内存的保有量随之增加，易造成数据积压，具体负载度与内存的状态如图 5 所示。

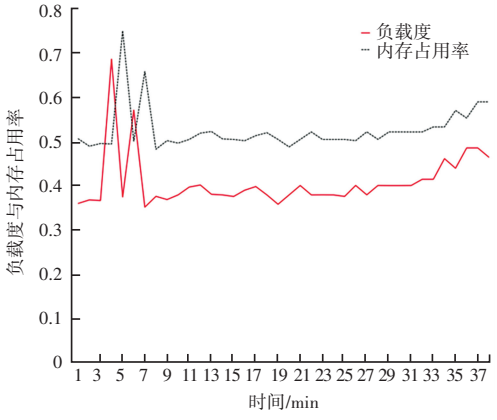

图 5　负载度与内存状态

Fig. 5　Memory status with load level

弹性缓存管理系统会综合负载报告判断是否发生数据积压，若需要均衡则管理模块会读取当前的资源信息表，做出微调后将调整信息发送给协调器，完成均衡。利用 Azure 自带的缓存监控和网络监控可以验证该系统的可用性，内存状态如图 6 所示。

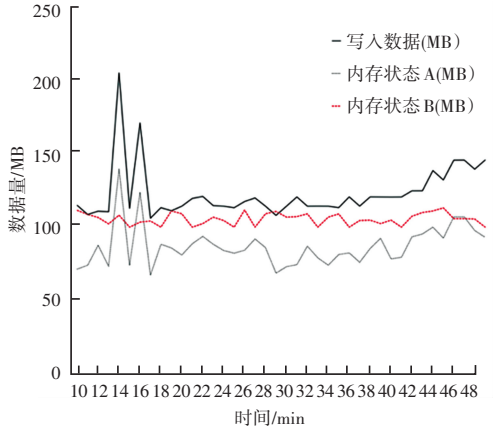

图 6　内存状态图

Fig. 6　Memory status

在图 6 中，写入数据为上游一个周期写入内存的数据量，内存状态 A 为对照组未应用数据反压操作的内存状态，内存状态 B 为应用负载感知单元完成双向数据反压的内存状态。将周期性流式数据的周期定为 1 min，即上游数据抓取数据的周期为 1 min，上游在一个周期内将数据按照协议打包数据流；内存状态 A 随着数据写入的变化而变化，且其变化率明显高于内存状态 B，表明弹性缓存管理模块具有较强的数据反压能力，并能合理平衡数据写

入造成的节点数据积压问题。性能指标 A 为对照组,性能指标 B 为实验组,对比结果如图 7 所示。实验组的性能指标明显低于对照组,此时节点负载度与吞吐率相对平衡。

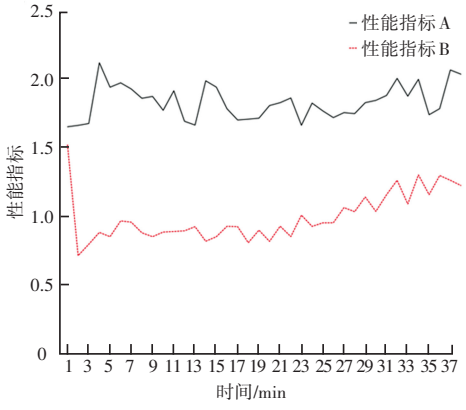

图 7 性能指标对比图

Fig. 7 Performance comparison

弹性缓存管理系统极大降低了缓存数据积压的风险。应用弹性缓存管理系统后,消除了高峰时期造成的高负载,缓存的利用率也得到提高,数据得到了及时处理。

4 结束语

为解决大型不稳定数据的缓存积压问题,本文设计了面向流式数据的弹性缓存管理系统。该系统整合了上下游数据流负载信息和缓存资源信息,并利用负载感知模型对数据负载进行量化分析,实时完成数据均衡。弹性缓存管理系统能够监控系统负载情况并进行预测,有助于及时调整系统资源,防止积压;在数据积压监测方面,利用负载报告提高了反数据积压的效率。下一步计划将该弹性缓存管理系统提升到更复杂的分布式架构中,包括引入分布式缓存迁移和缓存重定向等,以适应不同的分布式环境和应用场景。

参考文献

[1] WANG Y, JIN H, CHEN X, et al. Online-dynamic-clustering-based soft sensor for industrial semi-supervised data streams[J]. Sensors, 2023, 23(3): 1520.
[2] 乐涛, 陈庆奎, 黄陈. 面向恒压腹腔镜手术的云控制过程模型[J]. 智能计算机与应用, 2022, 12(9): 8-16,26.
[3] 李丽娜, 魏晓辉, 李翔, 等. 流数据处理中负载突发感知的弹性资源分配[J]. 计算机学报, 2018, 41(10): 2193-2208.
[4] 孙一佳, 丁箐, 徐云. 基于数据迁移策略的反压问题解决方法[J]. 计算机系统应用, 2022,31(5): 262-268.
[5] HANIF M, YOON H, LEE C. A backpressure mitigation scheme in distributed stream processing engines[C]//Proceedings of 2020 International Conference on Information Networking (ICOIN). IEEE, 2020: 713-716.
[6] 王绪亮, 聂铁铮, 唐欣然, 等. 流式数据处理的动态自适应缓存策略研究[J]. 计算机科学, 2020, 47(11): 122-127.
[7] 杨昊, 高全力, 李雪花, 等. 基于缓存价值的命名数据网络缓存优化策略[J]. 计算机与现代化, 2022(10): 95-99.
[8] 胡亚辉,朱宗卫,刘黄河,等. 面向任务调度优化的分布式系统信息管理框架[J]. 计算机系统应用, 2019, 28(11):9.
[9] 张凯, 车漾. 基于分布式缓存加速容器化深度学习的优化方法[J]. 大数据, 2021, 7(5): 150-163.
[10] 董文菁, 温东新, 张展. 基于 Alluxio 远程场景下缓存策略的优化[J]. 计算机应用研究, 2018, 35(10): 3025-3028.
[11] 魏占辰, 黄秋兰, 孙功星, 等. Alluxio 数据随机访问方法的研究[J]. 计算机工程与应用, 2020, 56(13): 77-83.
[12] 熊安萍, 朱恒伟, 罗宇豪. Storm 流式计算框架反压机制研究[J]. 计算机工程与应用, 2018, 54(1): 102-106,139.
[13] 陆佳炜, 吴涵, 陈烘, 等. 一种基于动态拓扑的流计算性能优化方法及其在 Storm 中的实现[J]. 电子学报, 2020, 48(5): 878-890.
[14] ZHAO F, LI S, ZHOU B B, et al. HCache: A Hash-based hybrid caching model for real-time streaming data analytics[J]. IEEE Transactions on Services Computing, 2018, 14(5): 1384-1396.