计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190800093

流式数据处理的动态自适应缓存策略研究

王绪亮[1]　聂铁铮[1]　唐欣然[2]　黄　菊[1]　李　迪[1]　闫铭森[1]　刘　畅[1]
1 东北大学计算机科学与工程学院　沈阳 110169
2 东北大学软件学院　沈阳 110169
(fracturesr@foxmail.com)

摘　要　在现代大数据处理应用场景中,流数据处理技术的应用十分广泛。消息中间件或消息队列常在流数据处理中起到数据缓冲的作用。Apache Kafka 常被用作数据缓冲中间件,Kafka 的工作性能在很大程度上决定着应用系统整体的性能。在实际应用中,Kafka 的上游数据源所产生的数据流量通常是不稳定的,静态的缓存策略不能适应这种多变的生产环境。针对这一问题,如果存在一种策略能根据上游流量变化动态调整数据缓存,就能增强系统对环境的适应能力,实现流数据缓存处理的实时性和吞吐量性能的提升。动态缓存策略采用对上游数据流量监控的方法,通过使用 ARIMA 模型对未来流量进行预测,提前调整流数据存储转发设置。流数据缓存设置参数的最佳值来源于在各压力下对中间件系统性能进行实验得到的结果的多目标优化。对比实验结果证明,在流数据高峰到达期间,策略在保证一定最大延迟的前提下可以使 Apache Kafka 的数据缓冲吞吐量性能提高 150%以上,从而提高了系统的整体性能。

关键词:Apache Kafka 平台;时序预测;多目标优化;流数据处理;消息中间件
中图法分类号　TP311

Study on Dynamic Adaptive Caching Strategy for Streaming Data Processing

WANG Xu-liang[1],NIE Tie-zheng[1],TANG Xin-ran[2],HUANG Ju[1],LI Di[1],YAN Ming-sen[1] and LIU Chang[1]
1 School of Computer Science and Engineering,Northeastern University,Shenyang 110169,China
2 College of Software,Northeastern University,Shenyang 110169,China

Abstract　In current scenarios of the big data processing application, the streaming data processing technique is widely used. Message middleware or message queue is usually applied as the data buffer in streaming data processing. Apache Kafka is often used as the data buffer middleware. The performance of Kafka largely determines the overall performance of the application system. In practical applications, the streaming data generated by upstream data sources is usually unstable, and the static data caching strategy cannot adapt to this variable production environment. In view of this problem, if there is a strategy that can dynamically adjust the data cache according to the upstream traffic changes, the adaptability of the system to environment can be enhanced, the real-time processing of streaming data caching can be realized and the throughput performance can also be improved. In the dynamic caching strategy, a method of monitoring the upstream data traffic is proposed, and the ARIMA model is used to predict the future traffic of data streaming, so as to adjust the settings of streaming data storage in advance. The optimum setting parameter of streaming data cache comes from multi-objective optimization of the experimental results of middleware system performance under various pressures. Comparative experimental results show that, during the peak period of streaming data, the strategy can improve the throughput performance of Apache Kafka by more than 150% while guaranteeing a certain maximum delay, thus the overall performance of the message middleware system can be improved.

Keywords　Apache Kafka, Time series forecast, Multi-objective optimization, Streaming data processing, Message middleware

1　引言

随着信息技术的不断发展,人类活动所产生的数据正在以超越几何级数的速度快速膨胀。一般可以用 4 个 V 对大数据的基本特点进行描述,即容量(Volume)、高速(Velocity)、多样性(Variety)和有价值性(Value)[1]。大数据的处理模式目前分为两种,即批处理和流处理。流式处理的数据源是实时流数据,应用极为广泛,但在系统的性能方面对实时性的要求较高。例如,现代大型网站往往需要在用户浏览的同时收集关于用户浏览和点击行为的大量数据,这些收集到的信息需要被立即分析处理以反馈到用户的浏览界面和收到的推送上。又如,随着物联网技术的普及,许多智能设备的通信和工作都用的是过流式数据处理技术的互联工作。再如文献[2]中提到,在现代天文学中,用于观测新的天文现象、验证物

到稿日期:2019-08-16　返修日期:2019-12-17　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。
通信作者:聂铁铮(nietiezheng@mail.neu.edu.cn)

理模型的超大型天文观测技术必须依赖流数据处理技术，来管理近乎实时产生的海量天文数据。总之，提升流式数据处理的速度、准确性和稳定性具有重要的实用意义。

流式数据处理的挑战主要在于流式数据的背压问题。当接收数据速度大于数据处理速度，却又不希望丢失数据时，就需要一个数据缓冲区来容纳这些已接收而又未被处理的数据。这种缓冲机制常常利用消息中间件(MOM)或消息队列(MQ)工具实现。常见的消息中间件有 IBM 的 MQSeries、Apache 的 ActiveMQ、BEA 的 RabbitMQ 等[3]。Apache Kafka 是这类消息系统之一，被广泛应用于搭建高通量的流式数据处理系统。Wang 等[4]基于 Kafka 设计并实现了自动备份的日志系统，Ichinose 等[5]结合 Kafka 和 Apache Spark，开发了实时的视频流分析处理框架，Michael 等[6]将 Kafka 应用于物联网技术中，利用 Kafka 的高吞吐量和数据持久化特性实现了大量边缘端的离线、在线数据交换。

然而，在系统中使用消息中间件并不意味着完全消除了数据丢失的风险。在实际应用中，消息中间件的上游数据源常常不稳定地输出流数据。在高峰时刻，数据产生的速度仍然可能高于中间件的吞吐能力，造成数据丢失。另一方面，上游数据的压力可能会对中间件系统本身的性能造成影响。对于 Kafka 来说，吞吐性能和消息延时与系统配置具有极强的相关性，这些配置在不同的数据流压力下有不同的性能平衡性。如果对这样的配置能够根据实际的上游数据流量进行动态的改变，就有可能进一步提升 Kafka 在实际应用场景下的整体性能。

针对上述问题，本文首先基于实验数据验证决定 Apache Kafka 生产者模块吞吐性能的关键配置随上游数据流量变动有不同的调优值。其次，本文提出了一种基于流量监控和 ARIMA 时序预测的 Kafka 生产者动态自适应策略。该策略首先对数据流进行速率采样，并通过 R 系统对未来流量进行预测，从而预先设定较优的配置参数值。为避免速率采样造成额外的开销进而降低应用性能，每个采样点还将与上次的预测值进行对比，在预测值与采样值相差不大时缩短采样时间和降低采样频次。最后，通过对比实验验证该策略在 Kafka 吞吐量和响应延迟方面的性能提升。

本文第 2 节介绍相关工作；第 3 节和第 4 节分别介绍动态自适应策略和相关模型设计；第 5 节介绍实验并分析实验结果；最后总结全文。

2 相关工作

关于消息队列的研究由来已久，作为流式数据的背压缓冲机制并不是消息队列系统的唯一作用。Zhou 等[7]详细介绍了消息队列技术在应用程序系统设计中的各种应用。

如何尽量减小流数据负载波动给流数据系统带来的性能损失一直是流数据处理领域的重要研究课题之一。目前，主要的应对方法是“延时调整”策略，即在处理性能与负载出现明显不匹配时再加以调整，故而不可避免地陷入了“调整滞后”和“调整颠簸”的问题。因此，部分学者针对“延时调整”提出了“预先调整”的策略。Li 等[8]按照“预先调整”的思想，通过对流数据负载的预测实现了对流数据处理系统资源的预先调度，从而确保了计算机资源的合理弹性分配，提高了流处理系统的平均性能。

Apache Kafka 是为了满足现代大数据应用数据量庞大、数据产生速度快、时效性强等需求而诞生的流数据处理系统之一。这些系统一般都采用分布式设计，具有良好的容错性和高可用性。Cui 等[9]详细介绍了现代分布式流处理系统的一些特征和应用。

近年来，与 Apache Kafka 相关的研究数量繁多，但涉及其性能优化的相关研究相对较少。Ichinose 等[5]从视频流实时处理框架的设计角度出发，在不同集群配置下对 Kafka 的性能进行测试，在视频流实时处理应用中找到了 Kafka 缓冲集群的一组最佳配置，并通过实验证明了该系统的瓶颈已经转移到单个 broker 的磁盘 I/O 上。

在流数据缓存性能优化研究方面，由于 Kafka 拥有繁杂庞大的配置系统，并且它的性能表现与许多配置项有着或强或弱的关联性，因此大部分性能相关研究都从此处入手。

Kleppmann 等[10]提出一个单分区主题的吞吐瓶颈主要在于网络带宽或者顺序硬盘读写的速度，因此通过增加分区数量并将它们布置在不同的 broker 上可以获得更高的性能。

此外，对 Kafka 进行可配置系统的性能建模也是对 Kafka 进行性能优化的方法之一。现有研究对复杂软件系统的性能建模问题的探讨较为完整，技术较为成熟。Wang[11]基于系统性能建模的相关研究，实现了对 Apache Kafka 的性能预测及性能优化。该文献首先采用专家经验与正交试验设计结合的方法，提取出与 Kafka 性能具有较大关联性的配置特征，再通过 LASSO 机器学习算法进行筛选，得出初步的性能预测模型，最后通过对特征相关性及特征边界问题的进一步探讨进行模型修正。该文献还设计并实现了基于遗传算法的最佳配置获取算法。

Apache Kafka 本身拥有专用的 Producer 吞吐性能测试脚本 kafka-producer-perf-test.sh，用于提供相应的测试参数，得到性能测试输出。通过多次控制变量的实验，就能选取得到一组较优配置。这样的方法简单易行，并有一定的实际效果。但是该脚本提供的测试数据流是一个平稳输出的数据流，并不能完全模拟实际应用时波动的上游数据源，从而不能解决本文开头提出的数据流波动造成的性能波动问题。另外，该方法的实验数据并未经过科学的数据处理，是从离散的实验配置中选择出来的较优配置，具有一定局限性。

综上所述，现有的关于 Apache Kafka 性能优化的相关方法和研究均局限于静态或稳定上游数据源的情况下。这些研究未将实际应用时波动的数据流对系统性能的影响纳入考虑。本文基于文献[8]、文献[10]和 Kreps 等[12]的研究，通过调整生产者关键配置优化 Kafka 生产者的吞吐性能，同时，为了应对实际生产环境中的流量变化情况，设计了监控-预测-调整-反馈的工作模式，使 Kafka 生产者在动态情况下达到吞吐量性能最优。

3 动态自适应缓存策略

具有动态自适应策略的 Kafka 体系结构设计如图 1 所

示，该结构能够提高自身的高峰背压能力。其设计主要在数据源与 Kafka 之间添加了 4 个模块：监控、预测、调整和反馈。

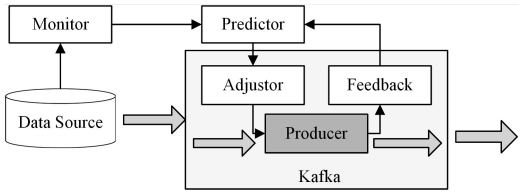


图 1 自适应策略体系结构图

Fig. 1 Architecture of Self-adapting strategy structure

监控模块负责采集测量从数据源流向消息中间件的数据流量，并将采集到的样本信息提交给预测模块。该样本信息主要包括数据流的瞬时速率以及采集该样本的时间。

预测模块负责对监控模块提交的样本信息进行整理，并加入预测模块维护的一个时间序列。之后，预测模块使用自回归移动平均模型（Autoregressive Integrated Moving Average model，ARIMA）对时间序列进行预测。该模型中，时间序列变量被认为是变量的过去观测值和随机干扰误差的线性函数[13]。ARIMA 模型拥有 3 个参数，p 参数表示预测结果是包含预测点前 p 个时间序列观测值的线性组合及误差项，q 参数表示预测结果是预测点前 q 个时间序列白噪声的线性组合及误差项，d 参数是使得时间序列平稳进行的差分次数。ARIMA 模型的一般数学描述如下：

$$z_t=\phi_0+\phi_1 y_{t-1}+\phi_2 y_{t-2}+\cdots+\phi_p y_{t-p}+e_t+\mu_0+\mu_1 x_{t-1}+\mu_2 x_{t-2}+\cdots+\mu_q x_{t-q}+\varepsilon_t \tag{1}$$

其中，ϕ_i 系列称为自回归系数；μ_j 系列称为平均滑动系数；y_m 系列是预测点的前 p 个时间序列；x_n 系列是预测点的前 q 个误差项；e_t 和 ε_t 均为误差系数。

预测模块得到的结果是对未来时刻上游数据流速的预测，这一预测结果被传入调整模块。调整模块使用内置的“流速-最佳 *batch.size*”映射提前获得未来时刻的最佳 *batch.size* 配置。调整模块使用的“流速-最佳 *batch.size*”映射来自使用带精英策略的非支配排序的遗传算法（Nondominated Sorting Genetic Algorithm II，NSGA II）对实验数据进行分析的结果。该算法是一种多目标进化算法（Multi-Ojbective Evolutionary Algorithm，MOEA）[14-15]。该算法的工作流程是首先产生一个随机的初始种群，然后将初始种群基于非支配排序。每个解都被分配一个与其非支配级别相等的适应度，适应度将被最小化。算法中引入了精英策略，父代产生的子代种群需要与父代竞争，通过将子代种群与以前发现的最佳非支配解进行比较，从而选出最优解。在产生最佳配置后，调整模块将对 Kafka 的配置进行调整，至此一次动态自适应过程完成。

反馈模块负责采集 Kafka 内部的吞吐量。采集到的信息可以用于反映动态调整效果或者作为参数参与调整。Kafka 通过 JMX 暴露了内部的运行状态，反馈模块通过 JMX 采集 Kafka 的内部信息。

总体来说，监控模块与 Kafka 本身串行工作，但与预测模块、调整模块和反馈模块并行工作，尽量使附加的动态调整策略对原系统的影响达到最小化。

4 面向 Apache Kafka 的动态自适应缓存策略实现

4.1 面向 Kafka 生产者的性能模型

Kafka 的生产者从数据源得到数据，并通过 send()方法完成发送，期间经过添加数据头、序列化、在 Accumulator 中打包，最终通过网络发送到实际存储数据的 broker 集群中。生产者的主要工作任务是对数据进行预处理并与 broker 进行网络 I/O。在软件层面，网络 I/O 是 Kafka 生产者性能最关键的影响环节。一般网络 I/O 的吞吐速率可以表示为：

$$delay=\frac{size}{bandwidth}+RTT \tag{2}$$

$$vel.=\frac{size}{delay} \tag{3}$$

其中，*size* 表示 I/O 数据的大小，*delay* 表示时延，*bandwidth* 表示带宽。

显然，吞吐速率和时延均与一次性传输的数据大小呈正比关系。因此，一般在网络 I/O 时将数据适度打包发送，能够在可接受的时延范围内获得较高的吞吐速率。同时，较大的打包意味着较少的发送次数，减少了对网络 I/O 的初始化开销，减轻了系统的工作负担。

对于 Producer 来说，一条数据从进入 send()方法开始到发送至 broker 为止的最大时延可以简单表示为：

$$delay=abatch.size+\frac{batch.size}{in.vel}+\frac{batch.size}{bandwidth}+RTT \tag{4}$$

其中，α 是比例系数，*in.vel* 表示上游数据进入 producer 的速率，*batch.size* 表示单次发送的打包量。上述变量是我们需要动态调整的关键配置。

式(4)假设上游数据进入 Producer 的速率不变，对进入生产者的数据进行序列化等预处理的时间复杂度是 $O(n)$，等待进入的消息将 *batch.size* 填满需要 *batch.size*/*in.vel* 时间，将积累完毕的消息构造成网络 I/O 请求的时间复杂度同样是 $O(n)$，这一项和预处理工作共同用 *abatch.size* 表示，最后两项即为常规网络 I/O 中的时延。

由该式可知，吞吐速率和时延同样均与一次性传输的数据量呈正比关系，然而，在环境因素参数 *in.vel* 增长时，Producer 时延表达式中的第二项将随之降低，此时可以在时延允许范围内适当提升参数 *batch.size* 值的大小，提高 Producer 的吞吐性能。

式(4)可以作为对 Producer 性能的简单描述，作为性能优化方向的指导和理论解释。然而，式(4)对 Producer 吞吐性能的评估与实际运行结果相比具有较大的误差。首先，式(4)中的 α 未知，其实际上也并不是固定值，而与数据的规模相关。其次，*in.vel* 表示数据进入生产者的速率，它应当是单位时间内进入系统的消息条数与消息的平均大小的乘积。在 *in.vel* 为定值时，如果单条消息的平均大小较小，则会频繁调用 Accumulator 的 append 方法，实际的吞吐性能会下降，可见其他外部因素如消息大小也可能是 Producer 吞吐性能的影响因素之一。

4.2 Kafka 动态自适应缓存工作流程

本文设计的 4 个模块的主要功能是通过监控上游流量和

时序预测，提前预置 batch.size 的大小，从而实现动态自适应功能。Kafka 动态自适应缓存工作流程如图 2 所示。

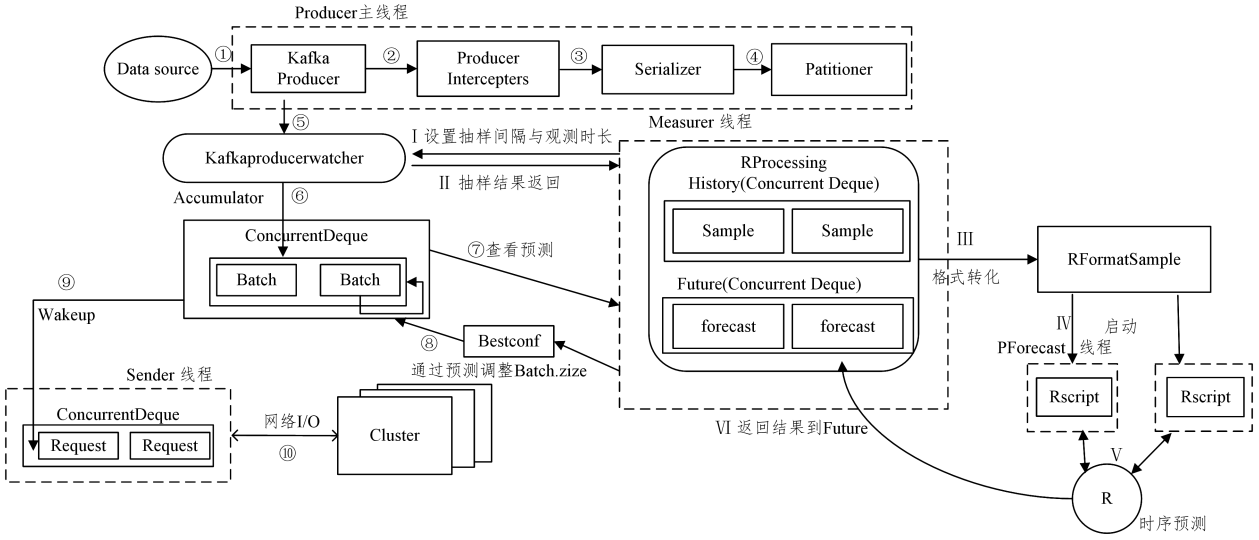


图 2 Kafka 动态自适应缓存策略工作流程图

Fig. 2 Dynamic adaptive strategy working flow path for Kafka

图 2 中，Measurer 是主要工作类，它在 Producer 主线程以外工作，负责协调监控模块和预测模块。为了最小化动态策略对 Kafka 本身的性能影响，Measurer 总是在进行一次测量后休息一段时间。在一个工作周期开始时，Measurer 首先从 RProcessing 中提取本次取样观察的时间和观测完毕后 Measurer 线程挂起休息的时间，然后打开开关，使由 send 方法向 Accumulator 发送的消息全部经过 KafkaProducerWatcher，KafkaProducerWatcher 以自身的一个原子长整形记录流过的数据量大小。在观测时间结束后，Measurer 关闭开关，将观测记录的数据流除以时间得到数据流速，并将其构造成时间序列样本点，加入 RProcessing 管理的历史时间序列。

RProcessing 首先将新的时序样本与上次的预测结果进行对比，设置下次 Measurer 的观测和休息时间。如果上次预测和测量结果相差较大，则下次观测时间长，休息时间短；反之，设置一个较短的观测时间和一个较长的休息时间。接着，RProcessing 新构造一个 RForecast 线程并将历史序列传入。

5 实验设置及结果分析

5.1 实验环境

实验在单个节点上进行，实验硬件环境为 CPU Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz，拥有 8 个核，内存 64GB。实验使用的软件环境中，操作系统是 Ubuntu 16.04.4 LTS，Kafka 版本是 2.1.0。

5.2 Kafka 生产者性能实验及结果分析

本文实验使用了 Kafka 发行版本中 Producer 的手动调优脚本 kafka-producer-perf-test.sh，该脚本模拟一个稳定的上游数据源通过 Producer 向 broker 匀速发送大小相同的消息。实验时，我们向一个单分区主题以不同速度发送 1 048 576 条大小为 1kB 的消息。对于同一发送速度，我们设置不同 *batch.size* 的值测量多次，分别记录吞吐速率和平均时延两个特征的平均值作为性能的衡量。上游压力下的实验结果如图 3 和图 4 所示。

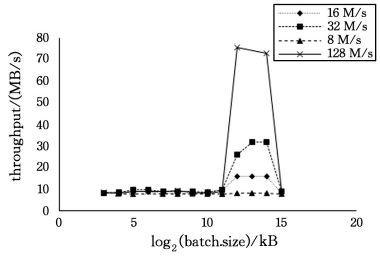


图 3 不同 batch.size 的 Producer 吞吐速率性能

Fig. 3 Kafka Producer throughput performance under different batch.size

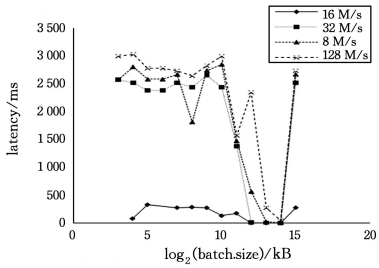


图 4 不同 batch.size 的 Producer 平均延时性能

Fig. 4 Kafka Producer average delay performance under different batch.size

由图 3 可以看出，对于一定速度以下的流量压力，*batch.size* 的改变并不影响 Producer 吞吐速率的大小；在阈值速度以上，Producer 吞吐速率-$\log_2$(*batch.size*)的图像在定义域内拥有一个较为明显的峰值。不同流量压力下的 Producer 吞吐速率性能图像形状相似，但在峰值处有较大差异。

从单条曲线来看，Producer 吞吐速率随打包量的增长而增长，这基本符合 4.1 节中的相关分析。然而，当打包量过大时，吞吐速率又会急剧减小。我们认为这是由于 Producer 用于打包的内存是从一个固定的内存池中分配出来的，过大的打包量设置使得每次构造新 batch 时会从内存池中拿走大量内存，导致内存池中可用的空间不足，如果此时又需要构造 batch，就要产生额外的等待时间，影响了吞吐速率。

从曲线间的关系来看，在一定范围内，流量压力越大，Producer 的吞吐速率越高。在 *batch. size* 较小时，每次分配的包很容易被填满，导致 Producer 频繁进行从内存池中分配并构造 batch 的操作，此时构造 batch 就成为了 Producer 吞吐速率的瓶颈，这使图像开始部分在各压力下 Producer 的吞吐性能非常接近；当 batch. size 较大时，分配构造 batch 的动作不会过多，此时 Producer 的吞吐速率就与上游数据压力的关联性较强；当 batch. size 过大时，内存池内存不足的问题成为吞吐性能的瓶颈，各曲线又趋于接近。

通过拟合，这一簇曲线的通用公式可以近似表示为：

$$(e^{x-A}+1)(0.012(x-2.9)(x-9.4)^2+0.2)(-e^{x-B}+1)+base \tag{5}$$

该式含有 3 个参数，即 A，B，*base*。A 与 B 一起确定了吞吐速率曲线的峰值和它对应的打包量，*base* 决定了吞吐速率曲线的下限。

从图 4 可以看出，在各种压力下，Producer 的平均延时性能表现是形状相似、并在纵坐标方向上递增的一族曲线。在压力达到某一程度后，延时基本不再升高。

从单条曲线来看，平均时延先增大后减小最后又增大，这似乎与之前对于 Producer 的模型分析相矛盾。事实上，当 *batch. size* 较小时，与对吞吐速率性能曲线的分析一样，Producer 频繁进行分配内存和构造 batch 的工作，这部分处理时延很大程度上抵消了打包量较小时，较小的积累数据包的等待时延和较小的发送时延优势；当 *batch. size* 过大时，内存池资源不足会造成额外的排队时延；只有当 *batch. size* 不太大也不太小时，Producer 不花费太多的时间构造 batch，不花费太多的时间进行网络 I/O，平均时延综合最小。

从曲线间的关系来看，一定范围内，压力越长，KafkaProducer 的时延越大，这主要是因为在单位时间由 send 送入 KafkaProducer 的消息数量随流量压力变多，需要处理的数据规模增大，但在数据流足够大时达到了 send 方法的瓶颈，时延就不再上升。

通过拟合，这一簇曲线的通用公式可以近似表示为：

$$(-e^{x-A}+1)(Cx)(-e^{x-B}+1)+base \tag{6}$$

式(6)含有 4 个参数，即 A，B，C，*base*。A，B 共同决定了极小值点处的下沉深度，C 决定了曲线的基本趋势，*base* 决定了平均时延的下限。

使用各压力下的吞吐速率拟合表达式和平均时延拟合表达式作为约束函数，在 PlatEmo 平台[16]使用 NSGA II 算法进行多目标优化，得到各压力下的 Pareto 非劣势解区间，取定义域中的点并进行拟合，结果如图 5 所示。

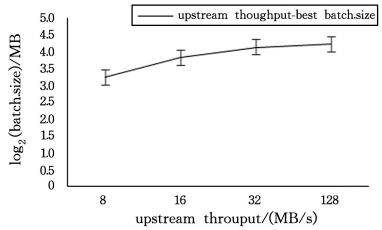


图 5 流量压力到最佳 $\log_2(batch.size)$ 映射曲线

Fig. 5 Data stream pressure to $\log_2(batch.size)$ curve

该图像即是一个“流量压力-最佳 *batch. size*”的映射。

5.3 自适应策略效果对比实验及分析

我们模拟实际生产应用环境中系统上游数据流压力陡增的情况，完成了对自适应策略效果的对比实验。

图 6 反映了数据压力激增时 Apache Kafka 的抗压缓冲性能。当某一时刻进入系统的数据较多时，Producer 来不及发送的数据就会形成“积压”，导致下一时刻只有较少的数据可以进入，因此进入系统的数据流瞬时速率变化呈“锯齿状”。这种“拉扯”将 Kafka 的平均缓冲性能钳制在 80 MB/s 左右。

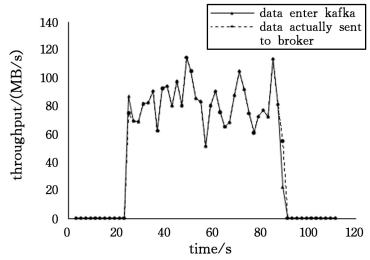


图 6 上游流量突然增大时 Kafka 系统的性能表现（未应用自适应策略）

Fig. 6 Kafka buffering performance under upstream data flow sudden increasing condition (without self-adapting strategy)

图 7 反映了数据压力激增时具备动态自适应策略的 Apache Kafka 的抗压缓冲性能。动态自适应策略对环境具有更强的适应性，在高峰时显著降低了“拉扯”，平均缓冲性能提高到 220 MB/s 左右。

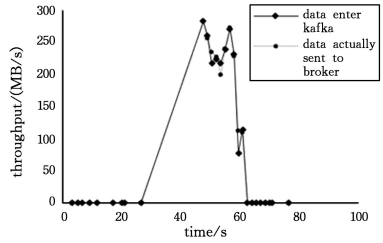


图 7 上游流量突然增大时 Kafka 系统的性能表现

Fig. 7 Kafka buffering performance under upstream data flow sudden increasing condition

对比实验验证了动态自适应策略可以显著提高 Apache Kafka 在高压下的缓冲性能，改进率为 175%。

然而，通过实验同时还发现，在非高峰时段，由于监控和预测的性能开销，应用动态自适应策略会使 Apache Kafka 的缓冲性能降低将近 1/4。图 8 和图 9 反映了这一现象。

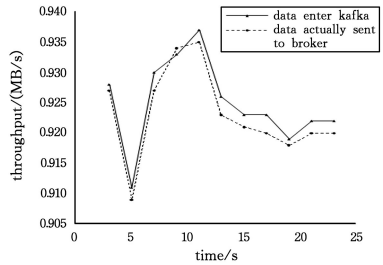


图 8 非高峰时段 Kafka 系统的性能表现(未应用自适应策略)

Fig. 8 Kafka buffering performance under reasonable data flow (without self-adapting strategy)

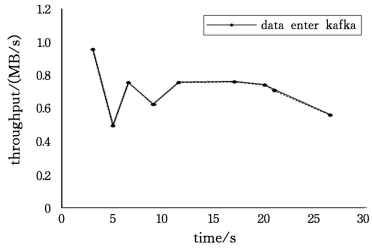


图 9 非高峰时段 Kafka 系统的性能表现(应用自适应策略)

Fig. 9 Kafka buffering performance under reasonable data flow (with self-adapting strategy)

结束语 提高流数据的高峰背压处理能力对各种计算机应用系统都具有重大的实用意义。本文通过提出一种根据上游流量压力动态调节缓存中间件配置的策略，使得 Apache Kafka 能够在变动的数据流压力下获得更好的平均缓冲能力。该策略调整 Producer 的关键配置，以使系统获得更高的性能。该动态调整策略和所采用的结构也可以应用于提高其他缓冲系统的性能。实验结果证明，该策略可以提高系统在高峰时段的缓冲性能。下一步将着手通过该策略动态调整更多的配置，进一步增强缓冲性能，并尝试解决低流量压力下动态调整策略影响性能的问题。

参 考 文 献

[1] LIU Y, WANG F, YANG M C. "Fast" and "Flexible" Big Data-Flexible Storage Technology in the Big Data Era [C] // 2015 Annual Meeting of the Information and Communication Network Technology Committee of the Chinese Communication Society. Beijing, China: Information and Communication Network Technology Committee of the Chinese Communication Society, 2015.

[2] YANG C, WENG Z J, MENG X F, et al. Astronomical Big Data Challenge and Real-time Processing Technology [J]. Computer Research and Development, 2017, 54(2): 248-257.

[3] LI L H, LI H Y, ZHANG F. Design and Implementation of Message Middleware[J]. Computer Engineering, 2000, 26(1): 46-48.

[4] WANG G, KOSHY J, SUBRAMANIAN S, et al. Building a Replicated logging system with Apache Kafka[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1654-1655.

[5] ICHINOSE A, TAKEFUSA A, NAKADA H, et al. A study of a video analysis framework using Kafka and spark streaming [C] // 2017 IEEE International Conference on Big Data. Boston, MA, USA: IEEE, 2017.

[6] MICHAEL D G, AZHARUDDIN V, GAURAV J, et al. Real-time Processing of IoT Events with Historic data using Apache Kafka and Apache Spark with Dashing framework[C] // 2017 2 nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT). Sri Venkateshwara Coll Engn, Bangalore, INDIA: IEEE, 2017.

[7] ZHOU S J, LIU J D, QIN Z G. Research on Message Queuing Technology: Review and an Example [J]. Computer Science, 2002, 29(2): 84-86.

[8] LI L N, WEI X H, LI X, et al. Elastic resource allocation for load burst sensing in stream data processing [J]. Journal of Computer Science, 2018, 41(10): 2193-2208.

[9] CUI X C, YU X H, LIU Y, et al. Overview of distributed flow processing technology [J]. Computer Research and Development, 2015, 52(2): 318-332.

[10] KLEPPMANN M, KERPS J. KAFKA, Samzaand the Unix Philosophy of Distributed Data [EB/OL]. [2019-08-01]. http://sites. computer. org/debull/A15dec/p4. pdf.

[11] WANG Z Y. Research and Implementation of Performance Modeling and Optimization Technology for Distributed Message System Kafka [D]: Xi'an: Xi'an University of Electronic Science and Technology, 2017.

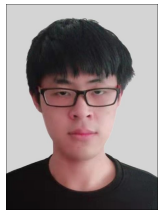
[12] KREPS J, NARKHEDE N, RAO N. Kafka: a Distributed Messaging System for Log Processing [EB/OL]. (2011-06-20) [2019-08-01]. https://www. microsoft. com/en-us/research/wp-content/uploads/2017/09/Kafka. pdf.

[13] ZHENG B Q, ZOU H X, HU X J. Research on Network Public Opinion Prediction Based on Turning Point [J]. Computer Science, 2018, 45(S2): 539-541.

[14] ZITZLER E, DEB K, THIELE I. Comparison of multiobjective evolutionary algorithms: Empirical results [J]. Evolutionary Computation, 2000, 8(2): 173-195.

[15] SRINIVAS N, DEB K. Multiobjective optimization using nondominated sorting in genetic algorithms[J]. Evolutionary Computation, 1994, 2(3): 221-248.

[16] TIAN Y, CHENG R, ZHANG X, et al. PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization [J]. IEEE Computational Intelligence Magazine, 2017, 12(4): 73-87.



WANG Xu-liang, born in 1998, bachelor. His main research interests include data mining and database.

NIE Tie-zheng, born in 1980, Ph.D, associate professor, master supervisor, is a member of China Computer Federation. His research interests include database, data integration and blockchain.