高技术通讯 2023 年 第33 卷 第10 期:1047-1059

doi:10.3772/j.issn.1002-0470.2023.10.005

基于时间序列模型的 Kafka 系统智能化管理方法①

周宇泽② 司鹏搏③ 张延华 李 萌 杨睿哲

(北京工业大学信息学部 北京 100124)

摘 要 区块链是分布式的数据存储系统,共识算法为区块链实现安全存储数据提供支撑和保障。Kafka 作为共识算法中的一种,其高吞吐速率、低时延的特点受到青睐。但使用 Kafka 算法的系统接受大量交易时,易产生数据倾斜,即分布式系统的多节点结构中,大量数据集中在少数节点,导致系统资源被占用、性能下降。为解决上述问题,本文提出基于时间序列模型长短期记忆网络(LSTM)的智能优化方法。通过学习过往生产者接收到的交易量,预测下一时刻面临的交易量,动态调整生产者节点数量,减少数据集中在少数节点的情况。实验结果显示,本文方法可以将 Kafka 系统时延降低 2~3 倍,吞吐速率提升 2~3 倍,与优化前相比系统效率提升 52.62%,比 2 种传统优化方法分别提升近 3% 和 40%,能耗仅小幅提升,系统使用情况保持更加合理。

关键词 区块链, Kafka, 长短期记忆网络(LSTM), 时间序列模型

0 引 言

近年来,随着互联网普及程度的不断提高和通信技术的高速发展,信息数据的安全问题引起了社会的广泛关注。区块链有着高度去中心化、不可篡改的特点,具有很高的安全性。目前,在各个领域中都进行着区块链的研究,例如车辆自组网(vehicle ad-hoc network, VANET)[1]、物联网(Internet of Things, IoT)[2]、智慧城市[3]和云计算[4]等。共识算法帮助区块链更好地完成节点间点对点通信、完成交易。比较有代表性的共识算法包括工作量证明(proof of work, PoW)[5]、权益证明(proof of stake, PoS)[6]、空间量证明(proof of space, PoSpace)[7]、实用拜占庭容错(practical Byzantium is fault-tolerant, PBFT)[8]以及近些年来逐渐被人们所熟知的 Kafka 共识算法。

Kafka 是一个高性能的分布式消息发布和订阅系统[9]。Kafka 中,Kafka Broker[9]指 Kafka 集群的服务器节点[10],每个 Kafka Broker 提供了名为主题的逻辑概念。主题[11]是存储消息的逻辑概念,被认为是消息的集合,同类型的消息会处于同一个主题中。生产者将消息以 push 的形式发送到 Kafka Broker,而消费者从 Kafka Broker 消费消息的过程是 pull,主动拉取数据,完成共识。

应用 Kafka 算法的区块链系统经常会面对海量交易等待处理的情况[12-13],大量交易集中在少数节点的情况,会导致系统性能下降[14]。针对 Kafka 系统中的消费者和分区,文献[15]和[16]分别提出消费者/客户端负载均衡方法和改进型 Partition 过载优化,来解决吞吐速率下降、中央处理器(central processing unit, CPU)使用率过高等问题。文献[17]提出基于抽样的自适应调优方法,通过优化生产者个数、缓存空间大小、消息大小、分区数等系统参数来提升系统吞吐速率并降低时延。先前的研究大多致力于从消费者、分区和系统参数的角度,在算法和系统内部添加机制来进行优化,但却往往忽略了生产者。而生产者节点一般同时担任排序节点[18]、背

① 国家自然科学基金(61901011),北京市自然科学基金(L211002)和北京市教育委员会科技计划一般项目(KM202110005021)资助。
② 男,1999 年生,硕士生;研究方向:区块链技术,共识算法,机器学习;E-mail: 2459537208@qq.com。
③ 通信作者,E-mail: sipengbo@bjut.edu.cn。
(收稿日期:2022-12-28)

书节点[18]和提交节点[19]的责任。因此生产者需要执行的进程十分复杂多样,大量数据集中在生产者的缓存区[20]中是 Kafka 系统性能吞吐速率下降、时延上升甚至出现宕机情况的主要原因。因此从生产者的角度进行系统优化也是研究的重点。

长短期记忆网络(long short term memory,LSTM)模型[21]从本质上讲是循环神经网络(recurrent neural network, RNN)的一种特定形式[22]。LSTM 模型以 RNN 模型作为基础模型,使用增加门限的方法来解决 RNN 模型中的短期记忆信息持久性不够高的问题。时间序列[23]分析是将某种现象某一个统计指标在不同时间上的各个数值,按时间先后顺序排列而形成的序列,旨在预测未来事件发展的趋势和规律。相比其他网络,LSTM 的优势包括:输入及输出属于时序数据,并要求全局化处理;输入和输出的元素级别对应的时间跨度大;数据长短适中。LSTM 应用的领域包括:文本生成、机器翻译、语音识别、生成图像描述、预测疾病、故障[24]和股票[25]等。

本文结合区块链节点的特点,提出基于 LSTM 时间序列的系统管理方法,在不过多修改 Kafka 算法内部机制的条件下,使系统更加智能化。利用 LSTM 时间序列模型可以胜任预测工作的特点,根据系统之前接收到的交易量,预测下一时刻即将收到的交易量,动态调整生产者的个数,既可以解决吞吐速率下降、时延上升等性能问题又不造成资源浪费,使系统资源利用处于相对合理区间。

1 问题建模

本节首先对问题及解决方案进行分析建模,而后建立整体系统优化模型。

1.1 Kafka 模型

目前常用的 Kafka 系统,以吞吐速率、时延、能耗、系统效率、使用率等指标来衡量系统的性能和合理性。系统的总吞吐速率与生产者吞吐速率以及分区关系为

$$T_T^t = P \times T_p^t \tag{1}$$

其中,T_T^t 为 t 时刻系统总吞吐速率,P 为分区个数,T_p^t 为 t 时刻生产者吞吐速率。

sum_t 表示 t 时刻需要处理的总交易量:

$$sum_t = Na_t + Nb_t + \cdots + Nn_t \tag{2}$$

式中,$Na_t + Nb_t + \cdots + Nn_t$ 是 t 时刻各用户向系统发送的交易量。

T_{pk_t} 为单个生产者的吞吐速率,设:

$$T_{pk_t} = \frac{\sum_{u=1}^{sum_t} b_m}{T_d} \tag{3}$$

其中,k_t 表示 t 时刻生产者数量,b_m 代表 t 时刻需要处理的单个消息大小,$\sum_{u=1}^{sum_t} b_m$ 表示 t 时刻系统需要处理的消息总大小,T_d 表示时延。

T_p^t 为 k_t 个生产者总吞吐速率:

$$T_p^t = \sum_{\omega=1}^{k_t} T_{pk_t} \tag{4}$$

理想状态下,每个生产者吞吐速率相等,由式(2)、(3)、(4)可得:

$$T_T^t = \sum_{\omega=1}^{k_t} \frac{P\sum_{u=1}^{sum_t} b_m}{T_d} = \frac{k_t P\sum_{u=1}^{sum_t} b_m}{T_d} \tag{5}$$

生产者、消费者、分区各占一部分系统存储空间。一般理想状态下,各生产者所占空间相等[25]。

Ω 代表机器(节点)数量:

$$\Omega = 2[V_{\max}(C_n)/w_t] + 1 \tag{6}$$

一般情况下 $\Omega \geqslant k_t > 1$,C_n 表示副本数量,w_t 为权重系数,$V_{\max}$ 表示生产者峰值生产速度。

根据式(6),在理想系统状态下,系统最大时延 T_d 可表示为

$$T_d = \frac{\sum_{\mu=1}^{k_t} M_{k_t}}{V_{\max} k_t} = \frac{c\sum_{u=1}^{sum_t} b_m}{V_{\max} k_t} = \frac{2cC_n\sum_{u=1}^{sum_t} b_m}{w_t k_t(\Omega - 1)} \tag{7}$$

其中,c 为数据的压缩比,$\sum_{u=1}^{sum_t} b_m$ 为 t 时刻系统缓存空间需要处理的交易总大小,设每个生产者缓冲区域大小为 M,$\sum_{\mu=1}^{k_t} M_{k_t}$ 为所有生产者缓存空间总大小。在各生产者缓存空间相同的一般系统状态下,可以推导出:

$$T_T^t = \begin{cases} \dfrac{P\sum_{u=1}^{sum_t} b_m}{T_d} & \sum_{u=1}^{sum_t} b_m < \sum_{\mu=1}^{k_t} M_{k_t} \\ \dfrac{P\sum_{\mu=1}^{k_t} M_{k_t}}{T_d} & \sum_{u=1}^{sum_t} b_m = \sum_{\mu=1}^{k_t} M_{k_t} \\ 0 & \sum_{u=1}^{sum_t} b_m > \sum_{\mu=1}^{k_t} M_{k_t} \end{cases} \tag{8}$$

其中，k_t 表示生产者数量，P 表示分区数量，$\frac{P\sum_{u=1}^{sum_t} b_m}{T_d}$ 表示 t 时刻单个生产者的吞吐速率。

M_l^t 为 t 时刻系统生产者平均剩余缓存区域大小：

$$M_l^t = \begin{cases} \dfrac{\sum_{\mu=1}^{k_t} M_{k_t} - \left(\sum_{u=1}^{sum_t} b_m - T_T^t\right)}{k_t} & \sum_{u=1}^{sum_t} b_m > T_T^t \\ \dfrac{\sum_{\mu=1}^{k_t} M_{k_t}}{k_t} & \sum_{u=1}^{sum_t} b_m \leqslant T_T^t \end{cases} \tag{9}$$

M_T^t 为 t 时刻系统生产者剩余缓存区域总大小：

$$M_T^t = \begin{cases} \sum_{\mu=1}^{k_t} M_{k_t} - \left(\sum_{u=1}^{sum_t} b_m - T_T^t\right) & \sum_{u=1}^{sum_t} b_m > T_T^t \\ \sum_{\mu=1}^{k_t} M_{k_t} & \sum_{u=1}^{sum_t} b_m \leqslant T_T^t \end{cases} \tag{10}$$

E_t 为每消耗单位能量处理的交易大小：

$$E_t = \frac{\sum_{u=1}^{sum_t} b_m}{Ek_t} = \frac{sum_t b_m}{Ek_t} \tag{11}$$

其中，E 为节点作为生产者时，每秒消耗的能量。

M_t 为生产者系统使用率：

$$M_t = \frac{\sum_{u=1}^{sum_t} b_m}{\sum_{\mu=1}^{k_t} M_{k_t} + T_T^t} \tag{12}$$

其中，$\sum_{\mu=1}^{k_t} M_{k_t}$ 为 t 时刻生产者缓存空间总大小。

W_t 为系统效率：

$$W_t = \frac{r_t T_T^t}{Ek_t} \tag{13}$$

其中，r_t 为比例系数，代表 t 时刻系统可承受交易量范围和用户发送总交易量范围之比。

1.2 LSTM 时间序列模型

找到生产者个数 k_t、用户发送交易量 sum_t 与各项性能、能耗及系统使用情况的关系后，如何确定生产者个数尤为重要。构建预测模型，如图 1 所示。

模型学习先前用户发送的交易量，预测 t 时刻各用户发送的交易量和系统需要处理的总交易量 sum_t。

LSTM 模型主要包含了遗忘门、输入门、输出门与 1 个细胞单元。细胞单元状态可以控制信息传递给下一时刻。时间序列模型流程示意图如图 2 所示。

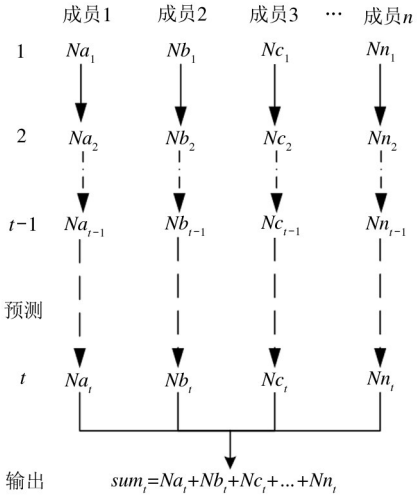

图 1　预测模型示意图

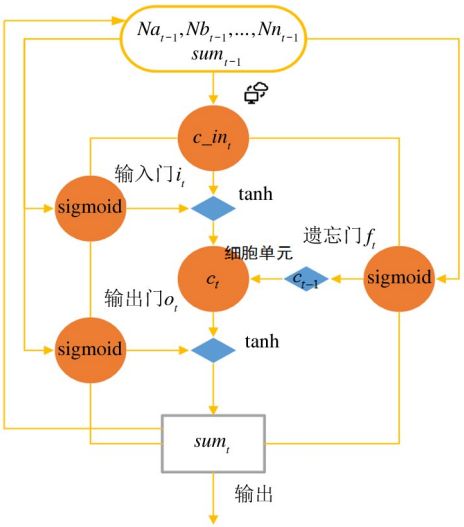

图 2　时间序列模型流程图

图中，c_t 是细胞单元，从下方输入 f_t、i_t、o_t 分别为遗忘门、输入门、输出门，用 sigmoid 层表示。sum_t 为输出，即预测到的交易总量。Na_{t-1}，Nb_{t-1}，…，Nn_{t-1} 为输入，即上一时刻用户发送交易量。c_{t-1} 为需要遗忘的对预测无用的数据。2 个 tanh 层分别对应细胞单元的输入与输出。

1.3 LSTM 模型执行步骤

LSTM 结构中使用了 2 种激活函数[26]，分别为 sigmoid 函数与 tanh 函数，其表达式分别为

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{14}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{15}$$

第 1 步　决定遗忘细胞状态中的哪些信息。

遗忘门通过 sigmoid 层来筛选，只有符合预期结果的信息可以通过细胞单元，根据上一时刻的输出和当前时刻的输入产生一个 0 ~ 1 之间的 f_t 值，以此决定是否让上一时刻学习到的用户发送交易量信息通过或部分通过，0 表示完全舍弃，1 表示完全保留。

$$f_t = \sigma\{W_f[sum_{t-1},(Na_{t-1}, Nb_{t-1},\cdots, Nn_{t-1})] + b_f\} \tag{16}$$

其中，f_t 为 σ 网络激活层，sum_{t-1} 为预测到上一时刻输出的交易总量，Na_{t-1}，Nb_{t-1}，…，Nn_{t-1} 为上一时刻输入的各用户发送交易量，W_f 为遗忘门权重，b_f 为遗忘门偏执量。

第 2 步　产生需要更新的信息。

第 2 步为 2 个部分，第 1 部分中输入门通过 sigmoid 层决定哪些信息用来更新候选值，第 2 部分生成新的候选值 $\tilde{c}_t$，并评估其保存至细胞状态的可能性。通过 tanh 层来执行本部分，对输入端进行激活，对细胞单元执行更新操作，将 c_{t-1} 更新为 c_t。

$$i_t = \sigma\{W_i[sum_{t-1},(Na_{t-1}, Nb_{t-1},\cdots, Nn_{t-1})] + b_i\} \tag{17}$$

$$\tilde{c}_t = \tanh\{W_c[sum_{t-1},(Na_{t-1}, Nb_{t-1},\cdots, Nn_{t-1})] + b_c\} \tag{18}$$

其中，W_i、W_c 为权重，b_i、b_c 为偏执量。

对旧的细胞状态进行更新，通过遗忘门消除不需要的信息，然后得出候选值。

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t \tag{19}$$

第 3 步　决定模型输出。

通过 sigmoid 层来得到一个初始输出，而后缩放 c_t 的值至 -1 ~ 1 之间，再与得到的输出进行逐对相乘，得到最终的输出。

$$o_t = \sigma\{W_o[sum_{t-1},(Na_{t-1}, Nb_{t-1},\cdots, Nn_{t-1})] + b_o\} \tag{20}$$

$$sum_t = o_t \times \tanh(c_t) \tag{21}$$

sigmoid 函数的输出不考虑先前时刻学到的信息，tanh 函数对先前学到信息的压缩处理，起稳定数值作用，智能化系统整体模型如图 3 所示。

本文构建的生产者个数决策模型，学习前 $(t-1)$ s

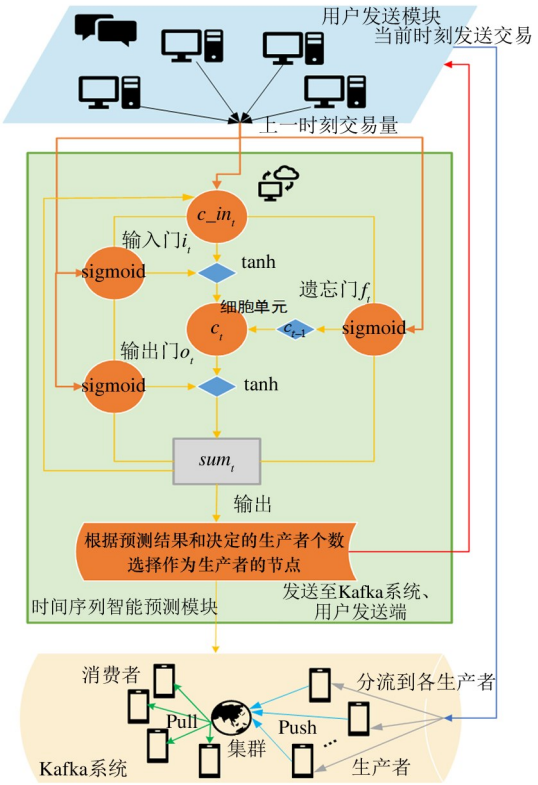

图 3　系统模型

用户发送交易的规律，通过时间序列模型 LSTM 算法使用仍然适用于本文模型的 sigmoid 函数与 tanh 函数完成预测第 t s 用户发送的交易量的工作，输出下一时刻选定的生产者个数，对生产者个数进行动态调整，在保证系统性能的情况下，达到降低能耗、合理利用系统资源的要求。

2　仿真实验

仿真环境建立在英特尔 11th Gen Intel(R) Core(TM)i5-1155G7@2.50 GHz 处理器。机带 RAM 16.0 GB 64 位操作。软件环境是在 Win 10 操作系统下运行 Python 3.6、Matlab 2022 以及 Jupyter Lab 平台。

分别设置模型学习率为 0.01、0.05，并结合实际应用中 Kafka 系统一般接收到的交易量，将样本数量设置为 50 个、100 个、150 个进行对比实验。

进行 500 回合的对比实验训练时，由图 4 中十字标记曲线可知，当学习率设定为 0.05、样本数量确定为 100 个时，系统误差最小，最低可降到 0.0124，收敛于 0.002 左右。因此 LSTM 模型学习率最终确定为 0.05，样本数量选定为 100，如表 1 所示。

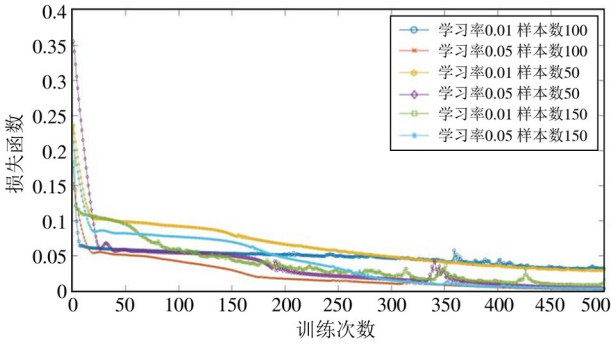

图 4　模型参数对比

表 1　LSTM 参数设置

参数名称	参数大小
训练次数	500,1000,3000,10 000
LSTM 层数	3
特征维度	7
神经元个数	16
Batch _ size	32
学习率	0.05
优化器	Adam
损失函数	MSE
训练样本数	100

后续实验中,基于时间序列 LSTM 算法的 Kafka 优化方法设置详细步骤如算法 1 及图 5 所示。

算法 1　基于时间序列 LSTM 算法的 Kafka 优化方法

1: 读取数据

2: 设置隐藏节点个数(隐藏层层数),进行数据归一化,设置特征向量

3: 数据重置 train _ series←0

4: n_i ←Look _ back 设置预测个数,进行数据序列化,设置神经元个数、梯度阈值、初始学习率、学习因子

5: 设置输入层、输出层、训练回合数,对模型进行编译,建立预测数据的时间序列

6: **repeat**

7: Input x_i 与上一时刻输出 h_{t-1} 进行 concat 操作。遗忘门确定通过 cell 的信息

$$f_t = \sigma\{W_f[h_{t-1}, x_t] + b_f\}$$

8: 输入门通过 sigmoid 决定哪些值用来更新,生成新的候选值

$$i_t = \sigma\{W_i[h_{t-1}, x_t] + b_i\}$$

$$\tilde{c}_t = \tanh\{W_c[h_{t-1}, x_t] + b_c\}$$

9: 细胞单元状态进行更新

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$$

10: 得到 h_t 并输出

$$o_t = \sigma\{W_o[h_{t-1}, x_t] + b_o\}$$

$$h_t = o_t \times \tanh(c_t)$$

11: 预测到的即将发送的消息总量 sum_t

12: **if** 预测消息总量 (sum_t) ⩾ n_{k1}

return $k_t = k_1$

13: **else if** n_{k1} > 预测消息总量 (sum_t) ⩾ n_{k2}

return $k_t = k_2$

⋮

14: **else** n_{kn} > 预测消息总量 (sum_t)

return $k_t = k_n$

15: **end if**

n_{k1}, n_{k2}, …, n_{kn} 为预设的有关 k_t 取值的预测发送数据总量临界线

16: **end**

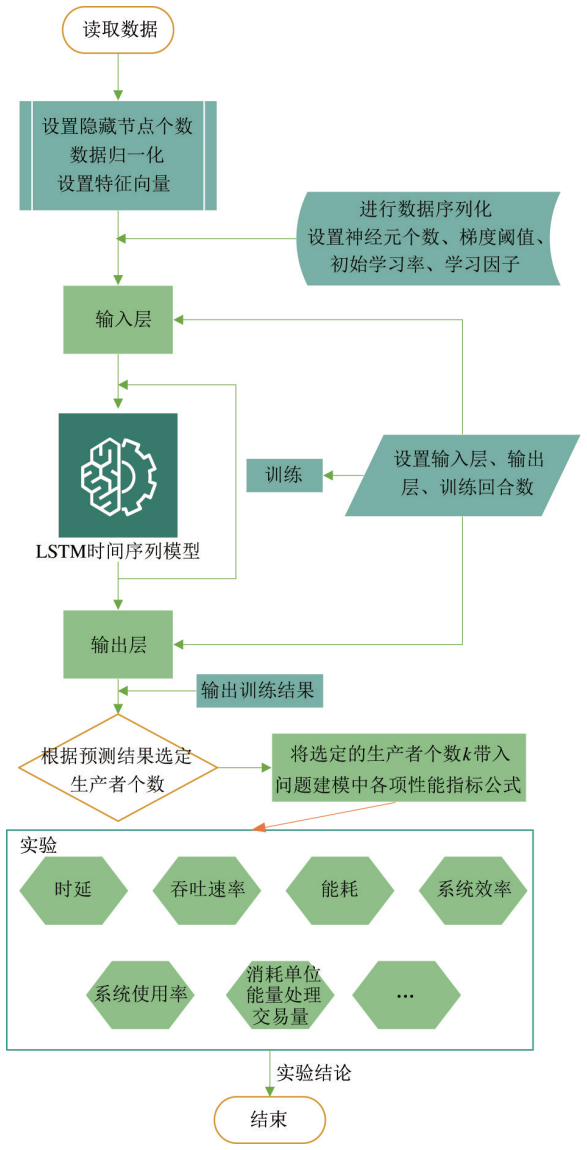

图 5　优化方法实验设置

交易数据集为随机生成,总体处于 1 ~ 100 个,通过 Excel 导入模型。模型参数设置如表 2 所示。

表 2　模型参数设置

名称	参数
成员个数	3
成员 1 发送交易量状态	3 ~ 100 个
成员 2 发送交易量状态	19 ~ 25 个
成员 3 发送交易量状态	15 ~ 45 个
生产者个数为 k_t *	1　2　3
参考值临界线	50　100
数据个数	100 100 100

注 * :为便于程序设计和实验,生产者个数选定为 1、2、3 后,用 k 表示。

时间序列模型执行步骤如下所示。

$$f_t^n = \sigma\{W_f^n[sum_{t-1}, Nn_{t-1}] + b_f^n\} \tag{22}$$

$$i_t^n = \sigma\{W_i^n[sum_{t-1}, Nn_{t-1}] + b_i^n\} \tag{23}$$

$$\tilde{c}_t^n = \tanh\{W_c^n[sum_{t-1}, Nn_{t-1}] + b_c^n\} \tag{24}$$

$$c_t^n = f_t^n \times c_{t-1}^n \times i_t^n \times \tilde{c}_t^n \tag{25}$$

$$o_t^n = \sigma\{W_o^n[sum_{t-1}, Nn_{t-1}] + b_o^n\} \tag{26}$$

$$sum_t = o_t^n \times \tanh(c_t^n) = Na_t + Nb_t + Nc_t \tag{27}$$

$$\begin{cases} k = 1 & sum_t < 50 \\ k = 2 & 50 \leqslant sum_t < 100 \\ k = 3 & sum_t \geqslant 100 \end{cases} \tag{28}$$

第 2 节公式中 n 取 1、2、3,分别设定 3 个序列模型。Na_t、Nb_t、Nc_t 分别对应成员 1、成员 2、成员 3 在第 t s 预测发送的交易量。通过总交易量所处区间,决定生产者个数。根据第 2 节中公式可得 LSTM 时间序列模型输出与 Kafka 系统性能关系,如表 3 所示。

表 3　LSTM 输出与 Kafka 性能的关系

名称	意义	相关项
T_d	时延	sum_t、k
T_T^t	系统吞吐速率	sum_t、k
M_l^t/M_T^t	生产者剩余缓存空间平均/总大小	sum_t、k
E_t	消耗单位能量处理的交易量	sum_t、k
M_t	生产者系统使用率	sum_t、k
W_t	系统效率	sum_t、k

使用 MinMaxScaler 数据归一化方法:

$$X_{\text{std}} = \frac{X - X.\min(axis = 0)}{X.\max(axis = 0) - X.\min(axis = 0)} \tag{29}$$

$$X_{\text{svaled}} = M_{\text{std}} \times (\max - \min) + \min \tag{30}$$

其中,$X.\min(axis = 0)$ 表示每列值的最小值组成的向量,同理 $X.\max(axis = 0)$ 表示每列值的最大值组成的向量。max 表示映射到区间的最大值,默认为 1。min 表示映射到区间的最小值,默认为 0。X_{std} 是标准化化结果,X_{svaled} 是 $\hat{Y}_i$ 归一化结果。

使用均方误差(mean square error, MSE)作为误差评价指标:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i) \tag{31}$$

式中,N 为样本数量,Y_i 是为真实值,$\hat{Y}_i$ 为模型预测值。优化器选择适应性矩估计(adaptive moment estimation, ADAM)的方法,对损失函数进行优化。

2.1　LSTM 预测

使用数据集分别进行 500、1000、3000、10 000 次训练。当训练次数达到 10 000 次时,误差率曲线在 0.025 以下收敛,最小可以达到 $(8.2807e)^{-4}$,损失函数如图 6 所示。

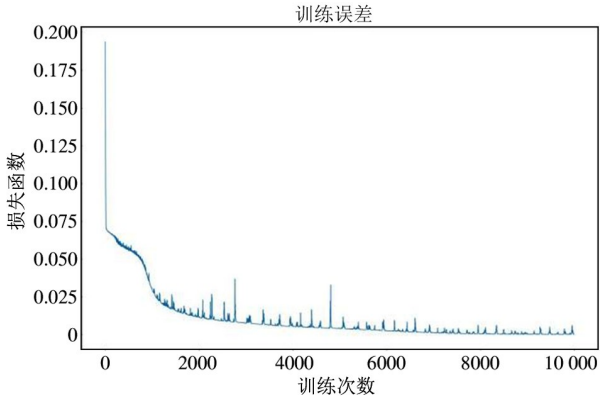

图 6　误差曲线

训练后得到预测下一时刻各用户发送的交易量,并根据交易总量决定 k 取值。结果如图 7 ~ 9 所示,在折线图上方显示系统经过预测后选定的生产者个数。

实验中会出现 $k = 1$、$k = 2$ 和 $k = 3$ 这 3 种结果。图中显示选定的 k 值,数据中包含需要通过遗忘门消除的对预测无用数据。用户1数据波动大,分布

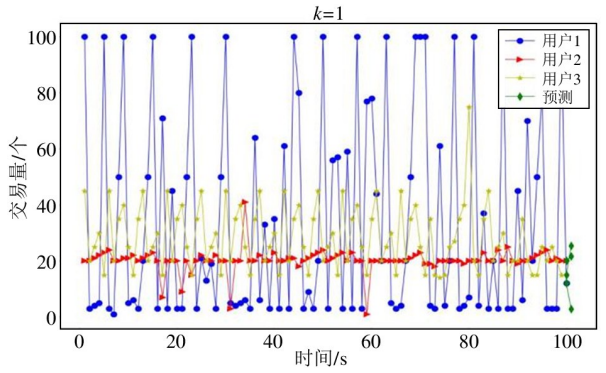

图 7　预测图 1

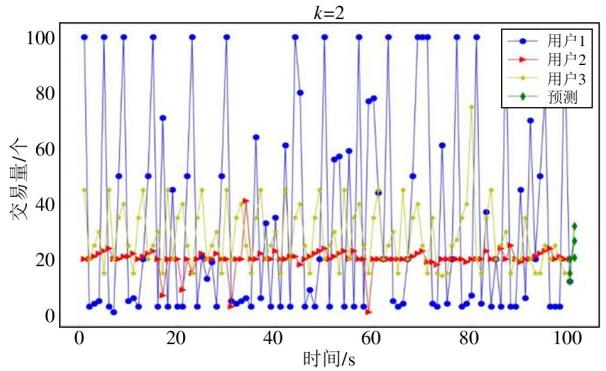

图 8　预测图 2

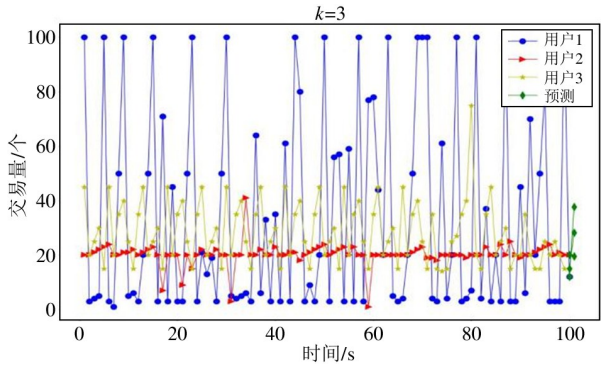

图 9　预测图 3

不均匀;用户 2 数据基本均匀分布在小范围内;用户 3 数据波动处于用户 1 与用户 2 之间,分布基本均匀。通过图 7 ~9 可以看出,对预测结果影响最大的是用户 1,用户 2 与用户 3 对结果影响不明显。

2.2　性能对比验证

根据前述公式及表 1 可知时间序列模型预测的消息总量、输出的生产者个数和 Kafka 系统性能的关系,根据时间序列预测结果及输出进行 Kafka 系统性能仿真实验。仿真实验按照预测模型,为 k 赋予了 1、2、3 取值,T_T^t 为系统吞吐速率,T_d 为时延。

设定数据的压缩比 c 设为 0.8,区块链中使用 Kafka 共识的机器数量 Ω 设为 5,生产者节点的个数 k 分别取 1、2、3。副本数量 C_n 设为 2,权重系数 w_t 设为 2,系统生产者缓存区内需要处理的交易总大小 $\sum_{u=1}^{sum_t} b_m$ 设为 0 ~ 110 kB,sum_t 为预测到的总交易量。处理交易量与时延的关系,如图 10 所示。

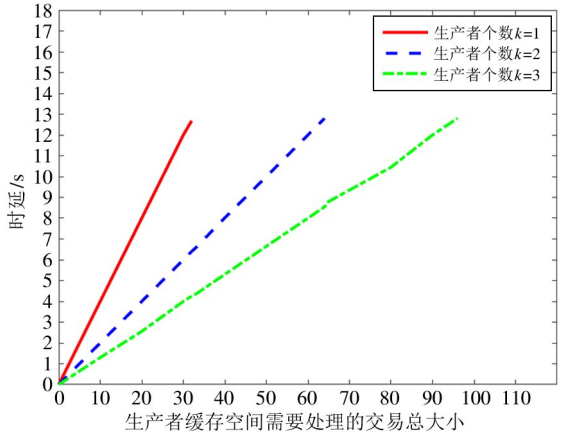

图 10　交易量与时延的关系

由图 10 可知,随着生产者同时需要处理的交易量的提升,系统时延也在逐步提高。通过使用 2 种激活函数的 LSTM 模型使生产者个数的增加,可以延缓系统时延的提高。当缓存区内需处理交易大小超过缓存区最大容量,引发系统宕机,时延图像停止变化。随着生产者个数的提升,系统承受能力显著上升。

设定分区数 P 为 20,k 分别取 1、2、3,数据压缩比 c 设为 0.8,每个生产者最大缓存空间 M 设为 32 kB。需要处理的交易总大小 $\sum_{u=1}^{sum_t} b_m$ 设为 0 ~ 110 kB,得出生产者缓存区处理交易量和系统吞吐速率 T_T^t 的关系,如图 11 所示。

由图 11 可知,使用包含 2 种激活函数的 LSTM 模型后,吞吐速率随着生产者个数增加而增加。当交易量小于最大缓存容量时,各曲线变化不大,只有轻微的波动。当需要处理的交易量大于生产者缓存空间容量时,引发系统宕机,吞吐速率归零。随着生产者个数的增多,最大容量提高,系统承受能力显著上升。

使用 LSTM 时间序列模型进行优化后,根据预测到的交易量增加了生产者个数。由图 11、图 12

对比实验可知，提升生产者个数确实可以提升系统性能，sigmoid 函数与 tanh 函数 2 种激活函数同样适用于本文模型，优化方向正确、方法有效。

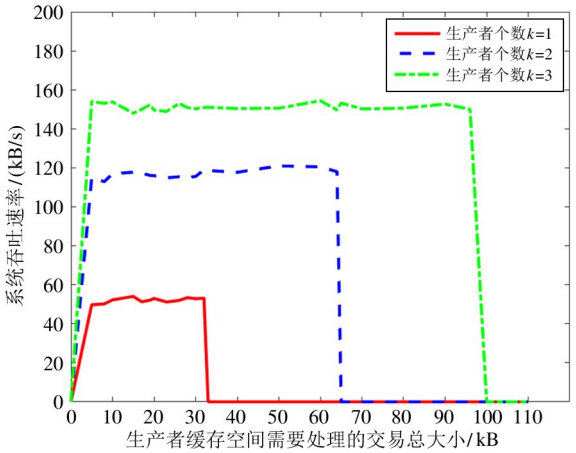

图 11　交易量与吞吐速率关系图

2.3　算法对比实验

为验证本文方法性能，分别采取 Kafka 原算法、本文方法、Kafka 中改进型 Partition 过载优化算法[16]和基于抽样的 Kafka 自适应调优方法[17]4 种方法相比较。

设 Kafka 原算法系统生产者个数 $k=1$，每个交易大小设为 1.6 kB，交易量范围为 0 ~ 150 个，副本数量 C_n 设为 2，权重系数 w_t 设为 2，设定分区数 P 为 20，数据压缩比 c 设为 0.8，每个生产者最大缓存空间 M 设为 32 kB。Kafka 中改进型 Partition 过载优化算法通过优化分区数量来提升吞吐速率，因此该方法分区数量 P_1 设为 25，生产者个数 $k=1$，其余参数与前文一致。

根据时间序列预测模型设定为

$$\begin{cases} k=1 & sum_t < 50 \\ k=2 & 50 \leqslant sum_t < 100 \\ k=3 & sum_t \geqslant 100 \end{cases}$$

如图 12 所示，随着交易量增加，Kafka 原算法、改进型 Partition 过载优化算法的时延曲线在本图中保持重合，且上升较快、系统承受力差，分区数提升未对时延产生明显影响。基于抽样的 Kafka 自适应调优方法将生者个数设定为 3 个时，系统时延低、承受能力强。本文方法通过预测接收到的交易量，动态生产者个数，随着接受到的交易量增多而增加生产者个数，可以有效降低时延且系统承受力与基于抽样的 Kafka 自适应调优方法持平。

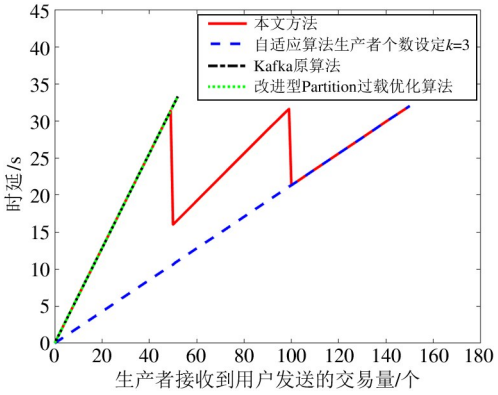

图 12　时延对比

如图 13 所示，Kafka 原算法的吞吐速率较低、系统承受力差。改进型 Partition 过载优化算法在一般情况下不对系统配置进行修改，仅在系统检测到即将过载时，通过提升分区数小幅提高了最大吞吐速率和系统承受力。基于抽样的 Kafka 自适应调优方法将生者个数设定为 3 个时，吞吐速率明显提升。本文方法通过预测接收到的交易量，随着接受到的交易量增多而增加生产者个数，可以有效提高吞吐速率和系统承受力。本文方法最高吞吐速率与基于抽样的 Kafka 自适应调优方法持平。

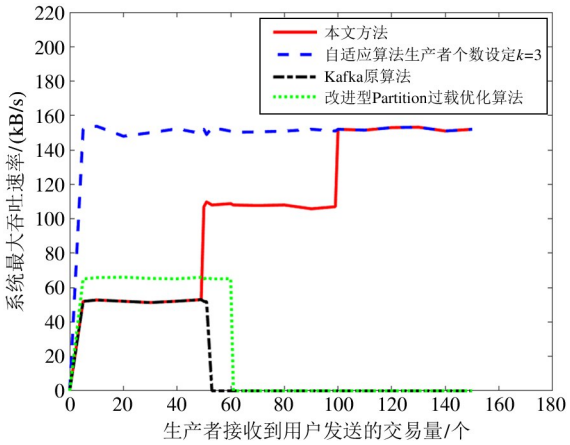

图 13　系统最大吞吐速率对比

根据式(9)及图 14 中数据计算 k 不同取值时的平均吞吐速率。当 $k=1$ 时平均吞吐速率为 52 kB/s，$k=2$ 时平均吞吐速率为 108 kB/s，$k=3$ 时平均吞吐速率为 151 kB/s，改进型 Partition 过载优化算法最大吞吐速率为 65 kB/s。单个生产者缓存

空间仍设为 32 kB。

由图 14 可知,根据预测下一时刻交易量,调整生产者个数以及提升分区数,均可以通过提升吞吐速率有效避免缓存空间过低和耗尽的情况。当预测到下一时刻交易过多可能导致缓存空间即将耗尽时,提升生产者个数,对交易进行分流,防止消息集中在某一个节点,生产者个数越多,缓存空间剩余越多。

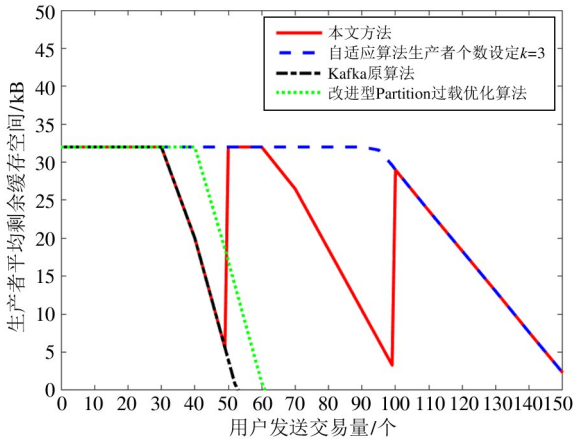

图 14　生产者平均剩余缓存空间

在实际使用中,系统能量损耗是必须考虑的方面,提升生产者个数固然可以提高系统性能,但需要处理的交易量较少时,多个节点执行生产者功能会造成能源浪费。设每台机器作为生产者时,每秒消耗能量 65 J,每个交易大小设为 1.6 kB。测试各方法所在系统的可接受用户发送消息量和消耗单位能量可处理的交易大小情况。

由图 15 可知,原 Kafka 算法在接收到的交易量较少时,消耗单位能量所处理的交易大小与本文方法持平,但其交易承受范围为 0 ~ 52 个。改进型 Partition 过载优化算法消耗单位能量所处理的交易大小先持平而后略高于于本方法,但其交易承受范围为 0 ~ 60 区间。基于抽样的 Kafka 自适应调优方法把生产者个数设定为 3 个,交易量承受范围为 0 ~ 150 个。在接受交易量较少时,消耗单位能量处理的交易量少,随着接受的交易量提升而提高,在实际应用中会造成大量的能源浪费。本文方法动态调整生产者个数,交易量承受范围也为 0 ~ 150 个,在交易量较少时本文方法高于基于抽样的 Kafka 自适应调优方法,交易量较大时与其持平,避免出现消耗能量高但处理交易量低的情况。

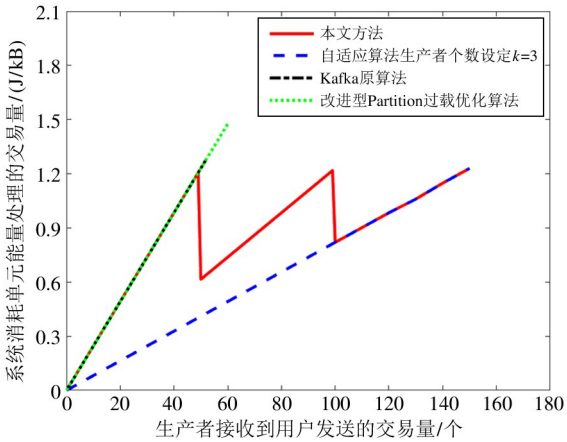

图 15　消耗单位能量处理的交易大小

由图 16 可知,原 Kafka 算法在接收到的交易量较少时,使用率与本文方法持平,但其无法承受较大交易量的情况。改进型 Partition 过载优化算法接收到的交易量较少时,使用率略低于本文方法,其系统承受能力较原算法有所提高但仍无法承受较大交易量的情况。基于抽样的 Kafka 自适应调优算法在发送交易量较少时,使用率远低于本算法。处理交易量较高时,效率与本文算法持平。当本文方法增加生产者个数时,系统使用率曲线先快速下降,而后随着交易量上升逐步提升。

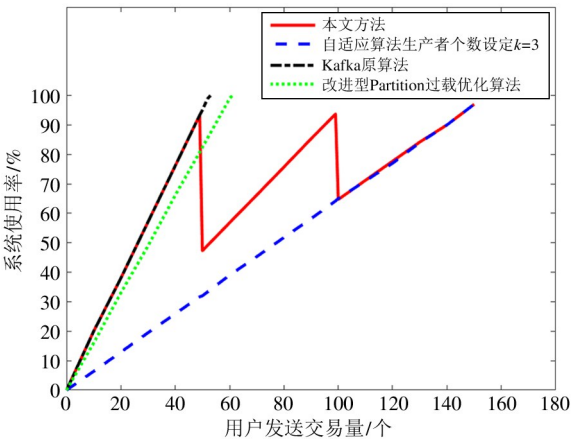

图 16　系统使用率

由图 17 可知实验中各方法在不同交易量情况下的能耗。Kafka 原算法可承受的交易量为 0 ~ 52 个,因此交易量大于 52 时,能耗柱状图归零。改进型 Partition 过载优化算法,可承受的交易量为 0 ~

60,因此交易量大于 60 时,能耗柱状图归零。

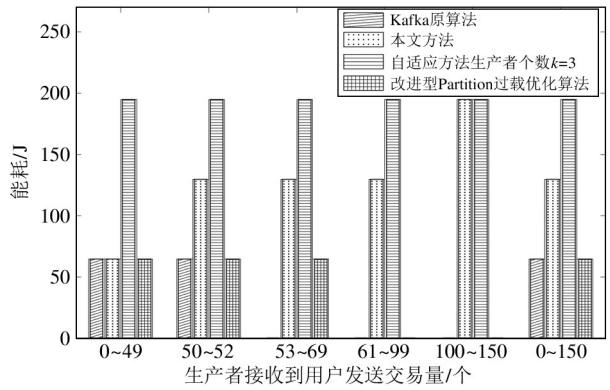

图 17　系统能耗

由图 18 可知,Kafka 原算法其系统使用率与能量之比最高、能耗低,但只能应对交易量较少的情况。改进型 Partition 过载优化算法可处理的交易量情况优于原算法,但其承受力远低于本文算法。基于抽样的 Kafka 自适应调优方法,将生产者个数设定为 3,可以应对各种交易量情况,但系统使用率与能量之比过低,在系统使用率低、处理较少交易时,依旧消耗大量能量。本文方法动态调整生产者个数,在交易量较少时系统使用率与能量之比较高,当预测到交易量上升时增加生产者个数后,系统使用率与能量之比有所降低,但可以应对各种交易量情况。在交易量处于 0 ~99 区间时系统使用率与能量之比高于基于抽样的 Kafka 自适应调优方法,在交易量处于 100 ~150 区间时 2 种方法持平。

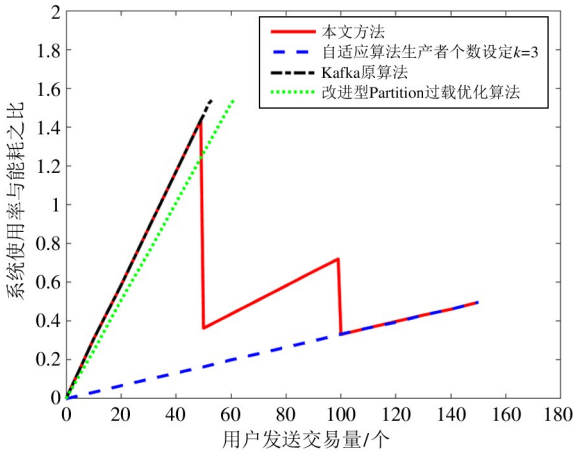

图 18　系统使用率与能耗之比

系统效率用于衡量系统对性能和能耗的兼顾情况。系统效率越高,意味着系统对两者的兼顾情况越好。根据式(13),计算各方法系统效率。

由图 19 可知根据不同生产者个数情况下平均吞吐速率和能耗计算结果。从实验中各种交易量情况进行整体分析,如图 17 最右侧柱状图所示,Kafka 原算法系统效率为 27.7%,改进型 Partition 过载优化算法系统效率为 39.96%,本文算法 3 种情况平均为 80.32%,基于抽样的 Kafka 自适应调优方法为 77.4%。原算法的能耗虽然低,但其吞吐速率和系统承载力同样较低,可接受的交易量范围只在 0 ~52 之间。改进型 Partition 过载优化算法由于只通过提升分区数而提升吞吐速率,并未增加生产者个数,因此其系统效率在系统可承受交易量范围内极高,但其只能处理 0 ~60 个交易的情况,系统承载力远低于本文算法。基于抽样的 Kafka 自适应调优方法,在将生产者个数设定为 3 后,虽然吞吐速率明显提升,但其能耗也大幅度增加,交易量较少时造成资源浪费。本文方法系统效率最高,系统承载力强且能耗较低。

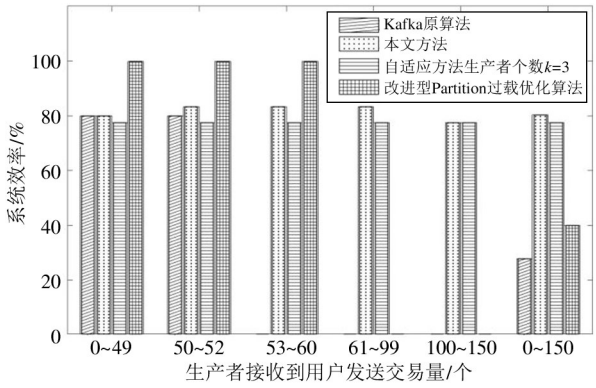

图 19　系统效率

实验中,式(10)可化简为

$$M_T^t = \begin{cases} Mk - (sum_t b_m - T_T^t) & sum_t b_m > T_T^t \\ Mk & sum_t b_m \leqslant T_T^t \end{cases} \tag{32}$$

设定 t 时刻系统开启 1 个生产者可以处理接收到的全部交易(即 1 个生产者 t 时刻的吞吐速率和缓存空间大小之和大于等于交易总大小),但系统中多于 1 个生产者的情况或类似情况会造成资源浪费。根据式(32)和对比实验中交易大小、交易量、

缓存空间大小、吞吐速率等数据进行计算。

由图 20 可知,Kafka 原算法、改进型 Partition 过载优化算法不存在资源浪费情况,但系统承受力低,可承受交易个数范围分别为 0 ~ 52、0 ~ 60。基于抽样的 Kafka 自适应调优方法将生产者个数设定为 3,可以提高系统承受力,可接受交易量为 0 ~ 150,但资源浪费比率高达 71.3%。本文方法系统可承受交易量为 0 ~ 150,且资源浪费比率远低于基于抽样的 Kafka 自适应调优方法,3 种情况平均为 6.66%。

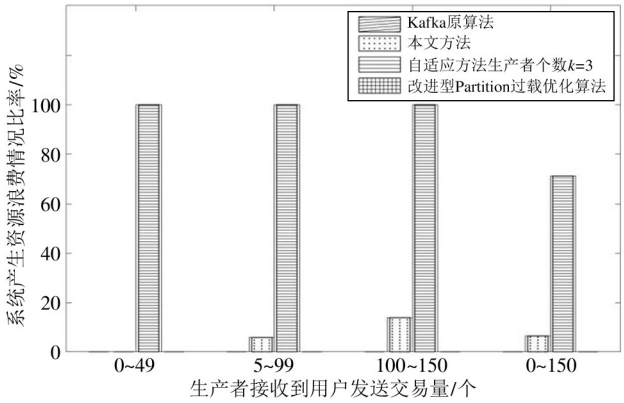

图 20　系统产生资源浪费情况比率

2.4　实验分析

本文方法可以在面对不同交易量情况下,智能化动态调整生产者个数,可将生产者个数由传统算法的 1 个提升为 2、3 个,使各项指标得到提升,综合评价情况达到最优。实验中,Kafka 原算法、改进型 Partition 过载优化算法、本文算法、基于抽样的 Kafka 自适应调优方法,其系统可承载交易量分别为 0 ~ 52 个、0 ~ 60 个、0 ~ 150 个、0 ~ 150 个,意味着 4 种算法分别在接收到交易个数为 52、60、150、150 时系统使用率趋近 100%,本文算法系统承载力大幅提高。实验中各方法最大吞吐速率分别为 52 kB/s、65 kB/s、151 kB/s、151 kB/s,本文方法将吞吐速率提升近 3 倍,时延降低了 2 ~ 3 倍。Kafka 原算法、改进型 Partition 过载优化算法、本文算法、基于抽样的 Kafka 自适应调优方法消耗单位能量所处理的交易最大分别为 1.28 kB、1.23 kB、1.23 kB、1.476 kB,本文方法在大幅提升系统承载力的基础上,消耗单位能量处理交易量与原算法和改进型 Partition 过载优化算法基本持平。且由图 15 可知,本文算法消耗单位能量处理交易量在绝大多数情况下高于基于抽样的 Kafka 自适应调优方法。由于提升生产者个数,因此系统能耗会出现提升,Kafka 原算法、改进型 Partition 过载优化算法、本文算法、基于抽样的 Kafka 自适应调优方法系统平均能耗分别为 65 J、65 J、130 J、195 J。本文算法在系统能耗指标方面并未达到最低,但由于 Kafka 原算法和改进型 Partition 过载优化算法能耗虽低,但其系统承受力同样低。且通过系统使用率与能耗之比可知,Kafka 原算法、改进型 Partition 过载优化算法、本文算法、基于抽样的 Kafka 自适应调优方法系统使用率与能耗之比最高分别为 1.538、1.538、1.453、0.497。基于抽样的 Kafka 自适应调优方法存在严重资源、能源浪费情况,因此需要结合系统效率做更加全面的评价。通过系统效率可知,本文方法系统效率为80.32%,相比于原算法的 27.7% 提升了56.62%,比改进型 Partition 过载优化算法的39.96%提升了40.63%,比基于抽样的 Kafka 自适应调优方法的 77.4% 提升了 2.92%。本文方法资源浪费比率为 6.66%,优于基于抽样的 Kafka 自适应调优方法的 71.3%,很好地兼顾了性能和能耗。因此综合性能、能耗及智能化考量,本文算法优于 Kafka 原算法、Kafka 中改进型 Partition 过载优化算法和基于抽样的 Kafka 自适应调优方法。

3　结　论

Kafka 系统在接受大量数据时,易产生数据倾斜的情况,从而导致性能下降的问题。本文提出了一种基于时间序列模型的智能化管理方法解决了上述问题。通过学习过往 Kafka 系统生产者接收到的交易量,预测下一时刻可能会面临的交易总量,动态调整作为生产者的节点数量,进而减少大量数据集中在少数节点的情况。实验证明,提升生产者个数可以在吞吐速率和时延方面大幅度提升系统性能,预测机制可以根据交易量灵活调整生产者个数,避免出现在低交易量情况下多个生产者运行,而造成能耗过高的情况,同时更加合理地利用系统资源,提高系统效率与合理性。本文方法为未来共识算法的优化提出新的思考,使针对 Kafka 系统的优化不再

停留于修改配置参数层面,而进入到使整个系统更加智能的全面优化阶段。为 Kafka 共识算法在区块链和其他领域中的应用提供新的思路和参考,使其可以更好地与大数据、物联网、机器学习、深度学习等领域协作,更全面地满足互联网时代对数据安全更加丰富且多元化的需求。

参考文献

[1] LU Z, LIU W, WANG Q, et al. A privacy-preserving trust model based on blockchain for VANETs[J]. IEEE Access, 2018, 6: 45655-45664.

[2] HUH S, CHO S, KIM S. Managing IoT devices using blockchain platform[C]//The 19th International Conference on Advanced Communication Technology. Pyeongchang: IEEE, 2017: 464-467.

[3] SHARMA P K, MOON S Y, PARK J H. Block-VN, a distributed blockchain based vehicular network architecture in smart city[J]. Journal of Information Processing Systems, 2017, 13(1): 184-195.

[4] SHARMA P K, CHEN M Y, PARK J H. A software defined fog node based distributed blockchain cloud architecture for IoT[J]. IEEE Access, 2018, 6: 115-124.

[5] NAIR P R, DORAID R. Evaluation of performance and security of proof of work and proof of stake using blockchain[C]//2021 3rd International Conference on Intelligent Communication Technologies and Virtual Mobile Networks. Tirunervili: IEEE, 2021: 279-283.

[6] GAŽI P, KIAYIAS A, ZINDROS D. Proof-of-stake sidechains[C]//2019 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2019: 139-156.

[7] ATENIESE G, BONACINA I, FAONIO A, et al. Proofs of space: when space is of the essence[C]//International Conference on Security and Cryptography for Networks. Cham: Springer, 2014: 538-557.

[8] 刘耀,金跃辉,崔毅东. 发布订阅系统中高效消息投递机制的研究[J]. 网络新媒体技术, 2015, 4(1): 15-23.

[9] 马跃,彭柏,韩大为,等. 基于 Kafka 集群的物联微服务数据接入模式的研究[J]. 信息技术, 2020, 44(12): 143-147.

[10] NOAC'H P L, COSTAN A, BOUGÉ L. A performance evaluation of apache Kafka in support of big data streaming applications[C]//2017 IEEE International Conference on Big Data. Boston: IEEE, 2017: 4803-4806.

[11] 王志泳. 分布式消息系统 Kafka 的性能建模与优化技术研究与实现[D]. 西安: 西安电子科技大学, 2017.

[12] 孙知信,张鑫,相峰,等. 区块链存储可扩展性研究进展[J]. 软件学报, 2021, 32(1): 1-20.

[13] 王锡亮,刘学枫,赵淦森,等. 区块链综述:技术与挑战[J]. 无线电通信技术, 2018, 44(6): 531-537.

[14] CHAI X C, WANG Q L, CHEN W S, et al. Research on a distributed processing model based on Kafka for large-scale seismic waveform data[J]. IEEE Access, 2020, 8: 39971-39981.

[15] 王郑合,王锋,邓辉,等. 一种优化的 Kafka 消费者/客户端负载均衡算法[J]. 计算机应用研究, 2017, 34(8): 2306-2309.

[16] 颜晓莲,章刚,邱晓红. Kafka 中改进型 Partition 过载优化算法[J]. 计算机技术与发展, 2020, 30(12): 88-91.

[17] 谢文康,樊卫北,张玉杰,等. ENLHS:一种基于抽样的 Kafka 自适应调优方法[J]. 计算机科学, 2020, 47(8): 119-126.

[18] XIE J, YU F R, HUANG T, et al. A survey on the scalability of blockchain systems[J]. IEEE Network, 2019, 33(5): 166-173.

[19] MAO C, NGUYEN A D, GOLAB W. Performance and fault tolerance trade-offs in sharded permissioned blockchains[C]//2020 IEEE International Conference on Blockchain and Cryptocurrency. Toronto: IEEE, 2020: 1-3.

[20] WU H. Research proposal: reliability evaluation of the apache Kafka streaming system[C]//2019 IEEE International Symposium on Software Reliability Engineering Workshops. Berlin: IEEE, 2019: 112-113.

[21] 滕金保,孔韦韦,田乔鑫,等. 基于 LSTM-Attention 与 CNN 混合模型的文本分类方法[J]. 计算机工程与应用, 2021, 57(14): 126-133.

[22] 栗然, 马涛, 张潇, 等. 基于卷积长短期记忆神经网络的短期风功率预测[J]. 太阳能学报, 2021, 42(6): 304-311.

[23] SHU X, ZHANG L, SUN Y, et al. Host-parasite: graph LSTM-in-LSTM for group activity recognition[J]. IEEE Transactions on Neural Networks and Learning Systems, 2021, 32(2): 663-674.

[24] 王鑫,吴际,刘超,等. 基于 LSTM 循环神经网络的故

障时间序列预测[J]. 北京航空航天大学学报, 2018, 44(4):772-784.

[25] 朱睿琪,孙丽,沈嘉和. 基于 LSTM 对股票走势的预测研究[J]. 信息技术与信息化, 2022(9):24-27.

[26] JO J, HWANG S, LEE S, et al. Multi-mode LSTM network for energy-efficient speech recognition[C]// 2018 International SoC Design Conference. Daegu: IEEE, 2018:133-134.

Intelligent management method of Kafka system based on time series model

ZHOU Yuze, SI Pengbo, ZHANG Yanhua, LI Meng, YANG Ruizhe

(Faculty of Information Technology, Beijing University of Technology, Beijing 100124)

Abstract

Blockchain is a distributed data storage system. Consensus algorithm can support and guarantee blockchain to achieve safe data storage. Kafka, as one of the consensus algorithms, is favored for its high throughput rate and low delay. However, when a system using the Kafka algorithm accepts a large number of transactions, it is prone to data skew, that is, in the multi-node structure of a distributed system, a large amount of data is concentrated on a few nodes, which leads to the occupation of system resources and performance degradation. To solve these problems, an intelligent optimization method based on long short term memory (LSTM) is proposed. By learning the trading volume received by producers in the past, the trading volume faced at the next moment can be predicted, and the number of producer nodes can be dynamically adjusted to reduce the data concentration in a few nodes. The experimental results show that the proposed method can reduce the delay of Kafka system by 2 – 3 times, increase the throughput rate by 2 – 3 times, and improve the system efficiency by 52.62% compared with the original Kafka system, which is nearly 3% and 40% higher than the two traditional optimization methods, respectively. The energy consumption is only slightly improved, and the system usage remains more reasonable.

Key words: blockchain, Kafka, long short term memory (LSTM), time series model