面向NVM的IoT时序数据多态协作压缩策略

蔡涛，雷天乐，牛德姣，戴健飞，黄泽宇，倪强强
江苏大学计算机科学与通信工程学院，江苏 镇江 212013

摘要
压缩策略是影响IoT时序数据存储系统性能的重要因素，而现有压缩策略缺乏针对NVM与IoT时序数据特性的优化机制。因此，提出了面向NVM的IoT时序数据多态协作压缩策略。首先，给出了IoT时序数据的组织结构。然后，针对IoT时序数据在一段时间内较稳定以及在用户态与内核态读写NVM适合的粒度差异较大的情况，设计了分层压缩策略。在用户态接收数据时，采用轻量级的数据压缩算法减少需存储的数据量，也减小了对IoT时序数据的存储效率的影响；针对IoT系统以查询和分析异常时序数据为主的特性，设计了深度压缩算法，在内核态对历史IoT时序数据进行深度压缩。其次，针对深度压缩历史IoT时序数据与存储新接收的IoT时序数据之间对NVM带宽的竞争，提出了写带宽保证的动态调整算法。最后，构建了面向NVM的IoT时序数据多态协作压缩策略原型PCCTSMS，并使用YCSB-TS工具进行测试与分析。实验结果表明，与InfluxDB、OpenTSDB、KairosDB和TVStore相比，PCCTSMS最高能提升161.3%的写吞吐率以及减少14.6%的存储空间。

关键词
数据压缩；IoT；时序数据；非易失性内存；存储系统

中图分类号：TP316 文献标志码：A doi: 10.11959/j.issn.2096-0271.2024048

A polymorphic cooperative compression strategy for IoT time series data based on NVM

CAI Tao, LEI Tianle, NIU Dejiao, DAI Jianfei, HUANG Zeyu, NI Qiangqiang
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China

Abstract
The compression strategy plays an important role in the performance of IoT time series data storage system. However, the current compression strategies can not adapt to the characteristics of NVM and IoT time series data. This paper proposes a polymorphic cooperative compression strategy for IoT time-series data based on NVM. Firstly, the overall structure of IoT time series data is given. Then, to address the consistent patterns in IoT time series data and the different granularity between user-space and kernel-space operations on NVM, a dual-compression strategy is devised. Initially, a lightweight compression method is applied directly as IoT time series data is received in user-space. This method efficiently reduces the volume of data for storage, while minimizing the impact on the timeliness of data storage. Moreover, a deep compression

algorithm is designed for the kernel-space, primarily focusing on querying and analyzing anomalous time series data. Additionally, to address the competition for NVM bandwidth between deep compression and data storage, a dynamic adjustment algorithm that guarantees write bandwidth is proposed. Finally, a prototype of the polymorphic cooperative compression strategy is implemented and YCSB-TS is used to evaluate the results. The results show that the proposed method can effectively improve the write throughput of IoT time-series data by up to 161.3% and reduce the storage space by up to 14.6%, compared with InfluxDB, OpenTSDB, KairosDB and TVStore.

Key words
data compression, IoT, time series data, non-volatile memory, storage system

0 引言

时序数据是按时间顺序产生的数据集合，物联网（Internet of things，IoT）时序数据是一种重要的时序数据。IoT系统通常包含众多IoT设备，而IoT设备的采集频率较高，产生的IoT时序数据的数据量大，因此对存储速度要求高。例如，新能源汽车的动力电池包含20多个采集间隔小于1 s的传感器，而一辆新能源汽车每天产生约30 TB数据，这给存储系统带来了很大的挑战。但对单个IoT设备来说，在一段时间内采集的数据通常相同或差异较小，如监护系统采集的健康人的血糖和血压值等。此外，IoT系统的数据的查询和分析主要针对异常值，删除历史数据中的正常值对系统的查询需求的影响较小。这些特性为设计新型IoT时序数据管理策略、提高存储和访问效率提供支持。

数据压缩是减少时序数据存储量和提高管理效率的常见方法。典型的数据压缩方法包括ZigZag、simple8b和游程编码等，但这些通用的数据压缩方法无法充分利用IoT时序数据的特性，存在压缩率低、计算开销大等问题。现有的时序数据存储系统通常通过删除某个时间点之前的历史数据来减少时序数据占用的存储空间，但这个方法会导致IoT系统无法跟踪和分析历史异常数据。

机械硬盘（hard disk drive，HDD）和固态硬盘（solid state disk，SSD）具备较大的存储容量，但读写速度有限且仅支持块的访问方式，难以满足IoT时序数据的存储和管理需求。非易失性存储器（non-volatile memory，NVM）具有接近动态随机存取存储器（dynamic random access memory，DRAM）的读写速度且支持字节寻址的访问方式[1]，能较好地满足IoT时序数据的存储需求，但相比于HDD和SSD，其存储容量较低，价格昂贵。因此，在满足IoT系统对数据查询和分析要求的前提下，如何减少IoT时序数据占用的存储空间是一个重要的课题。

NVM在用户态和内核态的读写特性差异较大，当读写粒度小于1 KB时，NVM在用户态的读写性能较高；当读写粒度大于1 KB时，在内核态的读写速度更高[2]。此外，NVM的并发读写带宽小于DRAM，在有多个并发读写任务时，带宽竞争问题更为显著[3]。当前的时序数据存储系统通常基于传统关系型数据库进行二次开发，其软件栈冗长，难以充分发挥NVM读写速度快的优势[1]；传统关系型数据库的存储优化策略通常针对HDD或SSD，缺乏对NVM中256字节原子写与字节级访问特性的支持。因此，面向NVM的IoT时序数据压缩策略具有较高的研究价值。

本文的主要贡献点如下。

● 针对IoT时序数据数据量大和数据值重复度高的特点，设计了多态协作的分级压缩策略。首先，在用户态接收数据时，采用轻量级的数据压缩算法减少需存储的数据量，也避免压缩算法的额外时间开销影响IoT时序数据的存储效率；然后，针对IoT系统以查询和分析异常时序数据为主的特性，在内核态对存储在NVM中的IoT时序数据进行深度压缩，保证IoT系统对异常时序数据的长期查询和分析能力。

● 针对深度压缩历史IoT时序数据与存储新接收的IoT时序数据之间对NVM带宽的竞争，提出写带宽保证的动态调整算法，在分解深度压缩操作的基础上，多层次地动态调整深度压缩的相关操作，在高效存储新接收的IoT时序数据的同时，完成对历史IoT时序数据的深度压缩。

● 在用户态基于PMDK（persistent memory development kit）实现用户态轻量级压缩算法，在内核态通过修改开源的英特尔傲腾持久内存的设备驱动实现深度压缩与写带宽保证的动态调整算法，构建了面向NVM的IoT时序数据多态协作压缩策略的原型PCCTSMS（polymorphic cooperative compression times series management system），并使用IoT时序数据存储系统专用测试工具YCSB-TS对原型系统的读、写吞吐率以及存储空间进行测试与分析。实验结果表明，与InfluxDB、OpenTSDB、KairosDB和TVStore相比，PCCTSMS最高能提升161.3%的写吞吐率以及减少14.6%的存储空间。

1 相关工作

1.1 IoT时序数据压缩算法研究现状

面对海量的IoT时序数据，数据压缩是减少存储空间的关键。TVStore[4]是一种基于Apache IoTDB的时序数据存储系统，它提出了时变压缩算法，该算法能够感知存储空间的大小，并能根据数据的存在时间判断其价值，从而选择不同的压缩策略。RRDtool是一个开源的时序数据存储系统，通过循环存储和定期合并策略，解决了大量数据的存储问题。RRDtool采用了轮询的压缩方法，其核心思想是维护一个固定大小的环形存储区域，当存储空间达到设定的上限时，新的数据将覆盖掉最旧的数据，从而形成一个环形的数据存储结构。然而，这种简单覆盖历史数据的方法，难以满足IoT系统对历史异常数据的查询和分析要求。InfluxDB采用了多种数据压缩技术来减少数据的存储空间占用，比如Snappy压缩、Gorilla压缩、Delta编码。Sprintz[5]是一种针对整型时序数据的压缩算法，在多个公开数据集上实现了高压缩比和3 GB/s的解压速度，低延迟和低内存开销的特性使其适用于边缘计算等资源受限的场景。PBE[6]采用动态规划的思想实现了对单调递增时序数据的压缩，但无法压缩一般的时序数据。CORAD[7]提出基于稀疏字典编码的时序流相关性感知压缩算法，利用多个时序之间的相关性消除冗余，提高压缩率，并通过阈值化控制机制尽可能地保留数据的原始信息。TS-NSM[8]是一种面向混合固态存储系统的IoT时序数据库，利用IoT时序数据与混合固态存储系统的特性，设计了一种融合删冗和压缩的数据存储策略，减少了存储空间占用，但并未充分考虑NVM的256字节粒度访问特性。

当前的时序数据压缩算法还未充分利用IoT时序数据变化小和以异常值查询为主的特性，也未充分考虑数据存储于NVM与HDD和SSD的差异，本文将利用以上特性对IoT时序数据的压缩进行改进。

1.2 NVM存储系统研究现状

NVM与HDD、SSD和DRAM存在较大差异，直接替换现有存储系统中的存储介质，难以发挥NVM的读写性能优势。MT²[3]是一个混合NVM和DRAM平台的带宽调节机制，通过硬件监控和软件报告等方式采集不同应用程序的内存带宽，设计了动态带宽节流算法，减少了带宽竞争对并发应用性能的影响。SplitKV[9]是一个混合NVM和SSD的键值存储系统，通过分析NVM和SSD在不同粒度下的读写性能，将不同大小的键值对存储到不同类型的存储介质中，在保证键值存储系统读写性能的同时，提高其经济性。此外，SplitKV还采用了FAST-FAIR B+树建立索引以提升查询效率，并使用热感知的键值迁移策略将冷数据从NVM迁移到SSD中。SLM-DB[10]是一个基于NVM的键值存储系统，通过融合B+树和LSM-tree提升其读写性能；将LSM-tree的多级结构转换为单级结构，并设计了选择性压缩机制来压缩数据，减少了LSM-tree的写入放大。KucoFS[11]是用户态和内核态协作的NVM文件系统，由用户态库和元数据内核态线程组成，利用协作索引、两级锁和版本化读取，将耗时长的任务从内核态转移至用户态，有效缓解了内核的瓶颈问题。OdinFS[12]是面向NVM的POSIX文件系统，采用内核代理线程来限制对NVM的并发访问，以缓解NVM缓存抖动问题。ROART[13]是面向NVM并针对范围查询优化的自适应基数树，使用数组统一管理Radix树的叶子节点，减少了查询的回溯次数；设计快速内存管理方式，防止内存泄漏，并引入即时重启策略以缩短数据恢复时间。

目前NVM存储系统的研究主要集中在优化读写性能、保证数据一致性、增强可扩展性和减少跨NUMA节点访问等方面，未充分利用IoT时序数据的数据量大、重复度高和以异常数据查询为主的特性。

2 IoT时序数据的组织结构

目前的时序数据压缩算法未充分利用IoT时序数据的特性，NVM在用户态和内核态的访问特性也存在较大差异，DRAM和NVM之间也存在带宽竞争等问题。因此，本文设计了基于NVM的IoT时序数据多态协作压缩策略。

IoT时序数据的结构（IoT-C）如**图1**所示。其中，max_val、min_val分别为正常IoT时序数据的最大值、最小值；l_blockAddr为轻量级压缩区的指针，d_blockAddr为深度压缩区的指针；dcmp_interval为深度压缩触发阈值；dcmp_time为上一次深度压缩的启动时间；dcmp_threshold1和dcmp_threshold2为深度压缩策略调节阈值。

为了减少压缩算法对IoT时序数据存储效率的影响，本文将数据压缩拆分为两个阶段，并对压缩算法的执行速率进行动态调节，设计了面向NVM的IoT时序数据多态协作压缩策略，其动态工作流程如

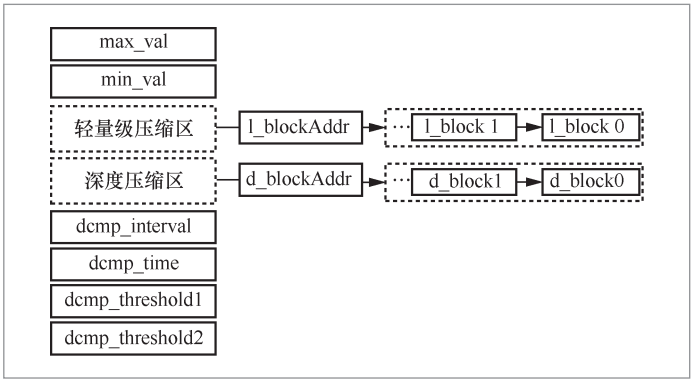

图1 IoT 时序数据的组织结构

图2所示。首先，在用户态接收IoT时序数据后，由用户态的轻量级压缩模块将其压缩成适合NVM存储的256 B轻量级压缩块并写入NVM，从而保证IoT时序数据的高效存储。其次，内核态中的深度压缩模块必要时会对历史轻量级压缩块进行压缩，构建深度压缩块后写回NVM，同时删除原有的轻量级压缩块，以减少占用的NVM存储空间。此外，写带宽保证的动态调整模块在分解深度压缩操作的基础上，对其进行多层次的动态调整，在完成历史IoT时序数据深度压缩的同时，保证存储新接收的IoT时序数据的效率。

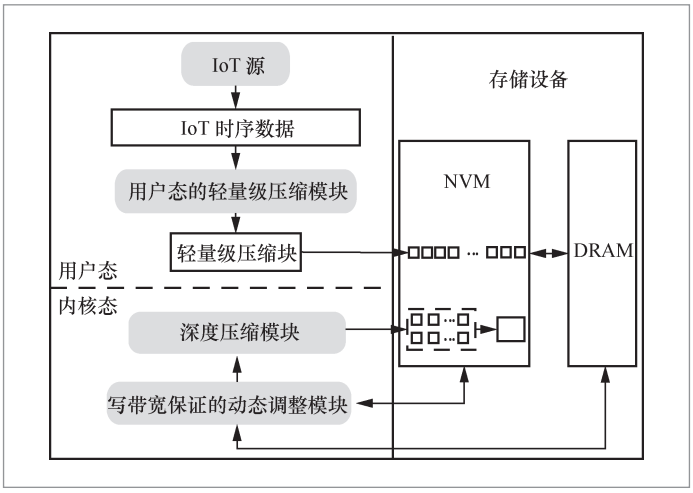

图 2　面向 NVM 的 IoT 时序数据多态协作压缩策略的动态流程

3 用户态的轻量级压缩算法

IoT时序数据通常由用户态应用程序接收，需要在较短时间内写入存储设备。为提高存储IoT时序数据的效率，在写入存储设备之前，通过压缩减少写入的数据量，但压缩数据会增加存储过程的时间开销。为保证存储IoT时序数据的及时性，笔者在用户态压缩数据，在压缩算法[8]的基础上进行改进，设计的轻量级压缩块的结构如图3所示。

首先，为轻量级压缩块增加e_flag属性。该属性用来标记该轻量级压缩块是否存储了异常IoT时序数据，以及异常IoT时序数据的分布类型。在构建轻量级压缩块时，使用式（1）判断IoT时序数据是否异常。若该轻量级压缩块未存储异常值，则将e_flag设置为0；否则，统计在该轻量级压缩块中是否超过半数的异常值连续出现3次及以上，若是，则将e_flag设置为2，若不是，则将e_flag设置为1。

$$D_x < \min\_val \| D_x > \max\_val \quad (1)$$

其次，将块首区的p_num属性由int类型修改为short类型，并将数据区的大

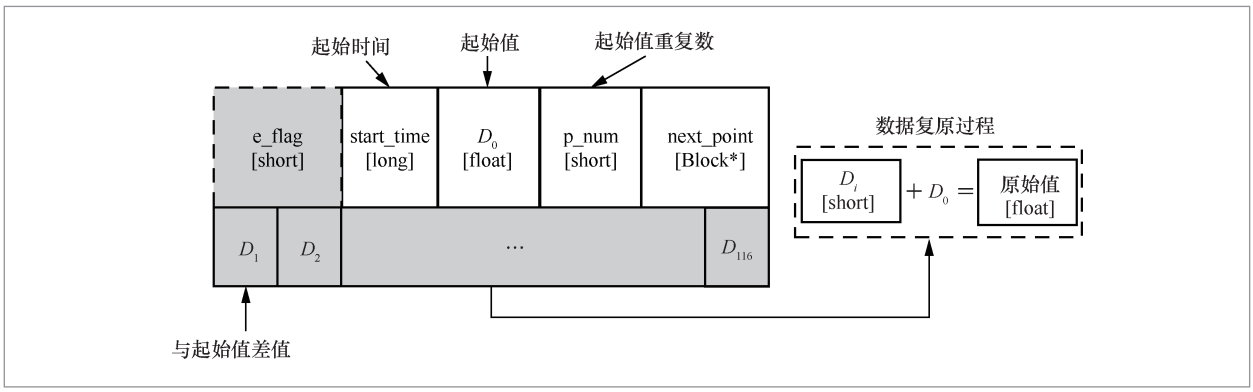

图 3　轻量级压缩块的结构

小减少为116个短整型数据的大小，将轻量级压缩块的大小调整为适合NVM特性的256 B。

用户态轻量级压缩算法主要通过数据块头部删冗和后部存储差值来减少需存储的数据量，并构建适合NVM存储的256 B轻量级压缩块；同时标记所构建的轻量级压缩块是否包含异常数据，为深度压缩提供支撑。具体的流程如算法1所示。

用户态的轻量级压缩算法能避免用户态和内核态之间的转换开销。使用轻量级方法压缩IoT时序数据，能够构造256 B轻量级压缩块，找出压缩块中的异常值并对相应的轻量级压缩块进行分类，为进一步压缩IoT时序数据提供支撑。

4 深度压缩算法

IoT时序数据具有数据量大、持续产生等特性，给NVM带来了很大的存储空间压力。同时，IoT时序数据的访问具有显著的时效性和差异性，一段时间后大部分IoT时序数据的访问频率非常低或完全无访问，但异常IoT时序数据值仍然会被访问。因此，IoT时序数据存储系统通常会对产生时间较长的IoT时序数据进行再压缩或直接删除。针对上述问题，本文设计深度压缩算法，对产生时间达到一定阈值的轻量级压缩块进行再压缩。

算法1：用户态的轻量级压缩算法

Input：IoT时序数据 , l_blockAddr

Output：更新后的l_blockAddr

```
1 if 系统刚启动或完成前一个轻量级压缩块的构建 then
2    start_time = firstIotTimestamp;//将首个接收到的IoT时序数据时间戳作为新轻量级压缩块的起始时间
3    D0 = firstIotValue;//将首个IoT时序数据值作为该轻量级压缩块的起始值
4    while IotValue == D0 do
5       repeatCount++;//若后续接收的数据值与D0相同则进行计数，直到出现值与D0不同的数据
6    endwhile
7    p_num = repeatCount;
8    checkOutlier(D0);//使用式（1）判断D0是否为异常的IoT时序数据
9    for 后续接收的IoT时序数据IotValue do
10      checkOutlier(IotValue);//使用式（1）判断是否为异常的IoT时序数据
11      storeDifference(D0, IotValue);//将后续所有接收的IoT时序数据与D0的差值以短整型的形式存储到数据区
12   until 当前轻量级压缩块数据区的数据量达到116个
13   setEflag(e_flag, IotValue);//标记该轻量级压缩块是否存在异常的IoT时序数据及其分布类型
14   next_point = l_blockAddr;//将轻量级压缩块的指针设置为轻量级压缩区的地址
15   完成该轻量级压缩块的构造，使用新产生的轻量级压缩块地址更新l_blockAddr;
16   return l_blockAddr;
17 endif
```

绝大多数轻量级压缩块并未存储异常的IoT时序数据，而在存有异常IoT时序数据的轻量级压缩块中，异常数据的分布呈现连续为主和离散为主等不同类型。对此，分别设计离散型和连续型深度压缩块，结构如**图4**所示。

两种深度压缩块首部都有e_flag属性，用于标识深度压缩块的类型，离散型和连续型深度压缩块的e_flag的值分别为1和2。

图4（a）给出了离散型深度压缩块的结构，前部与轻量级压缩块类似，start_time与end_time是该深度压缩块中IoT时序数据的起止时间戳；D_0是首个异常IoT时序数据的值；value_interval是包含1 015个短整型的时间间隔区，用来存放异常IoT时序数据之间的时间差；1 015个异常IoT时序数据与D_0的差值存放在深度压缩块的数据区；next_point是前一个深度压缩块在NVM中的地址。

连续型深度压缩块的结构如**图4（b）**所示，其大部分属性的含义与离散型深度压缩块相同，但不存在短整型的时间间隔区。数据区存储若干组连续出现的异常IoT时序数据，每组均包含g_time、g_num和异常数据区。与轻量级压缩块的构造类似，g_time是该组中首个异常值的时间戳与start_time的差值，用短整型表示，g_num是该组中异常IoT时序数据的数量，异常数据区存储的是异常IoT时序数据与D_0的差值，用短整型表示。

深度压缩算法的主要思想是，在内核态中，依据轻量级压缩块是否包含异常数据

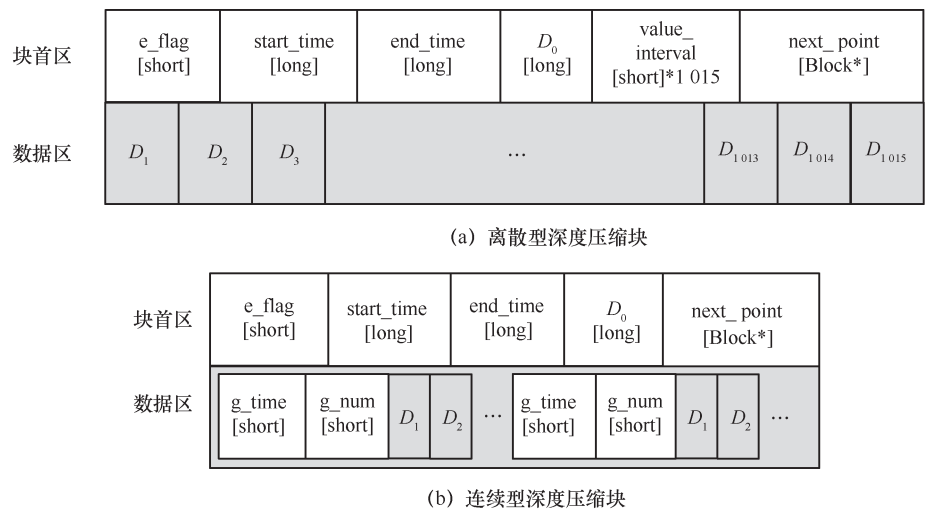

图4 离散型和连续型深度压缩块

算法2：深度压缩算法

Input：IoT时序数据，l_blockAddr，d_blockAddr，dcmp_interval，dcmp_time

Output：深度压缩区的地址d_blockAddr

1 if current_time - dcmp_time >= dcmp_interval then//距上次深度压缩的启动时间达到深度压缩触发阈值

2 dstart_time = current_time;//将当前时间戳记录为本次深度压缩的启动时间

3 while 轻量级压缩区中存在时间范围为（dstart_time，dcmp_time）的轻量级压缩块 do

```
4    if 轻量级压缩块lw_block的e_flag不为0 then
5      selectAsCandidate(lw_block);//将该异常轻量级压缩块作为深度压缩备选块
6    endif
7    if 深度压缩备选块candidateBlocks的数量达到16 then
8      readToBuffer(candidateBlocks);//在内核态中将所有深度压缩备选块读取至深度压缩缓冲区DcmpBuffer
9      start_time = firstOutlierIotTimestamp;//将产生时间最早的异常IoT时序数据时间戳记录到当前深度压缩块的start_time中
10     D0 = firstOutlierIotValue;//将其值记录到该深度压缩块的D0中
11     next_point = d_blockAddr;//将深度压缩块的指针设置为d_blockAddr
12     if 超过8个备选块的e_flag为2 then
13       e_flag = 2;//设置该深度压缩块的e_flag为2，构造连续型深度压缩块
14       storeContinuousOutliersInterval(start_time, g_time, IotTimestamp);//将该组连续异常IoT时序数据的起始时间IotTimestamp与start_time的差值保存到g_time中
15       storeContinuousOutliersNumber(g_num);//将连续异常数据的数量保存到深度压缩块的g_num中
16       storeDifference(D0, IotValue);//将剩余IoT时序数据与D0的差值保存至深度压缩块的数据区
17     else
18       dcmp_block.e_flag = 1;//设置深度压缩块的e_flag为1，构造离散型深度压缩块
19       storeDifference(D0, IotValue);//将后续异常IoT时序数据与D0的差值以短整型的形式保存到数据区中
20       storeDiscreteOutliersInterval(start_time, value_interval, IotTimestamp);//将每个异常IoT时序数据时间戳与start_time的差值保存到value_interval中
21     endif
22     if 完成一个4KB深度压缩块的构建 then
23       end_time = lastOutlierIotTimestamp;//将该深度压缩块中最后一个异常IoT时序数据的产生时间记录到end_time中
24       writeInKernalMode(dcmp_block);//在内核态中将新产生的深度压缩块写入NVM
25       updateAddr(d_blockAddr, dcmp_block);//用该深度压缩块的地址更新深度压缩区地址d_blockAddr
26       deleteLightweightBlocks(l_blockAddr, start_time, end_time);//删除该深度压缩块对应时间范围内所有的轻量级压缩块
27     endif
28   endif
29 endwhile//对范围内所有的轻量级压缩块完成了深度压缩
30 resetDeepCompressionBuffer();//重置深度压缩缓冲区
31 dcmp_time = dstart_time;//更新深度压缩启动时间
32 return d_blockAddr;//返回深度压缩块的地址
33 endif
```

以及异常数据分布类型，在仅保留异常数据的基础上，构造出不同类型的深度压缩块，并删除其他正常数据，从而在满足IoT系统查询和分析异常数据的同时，减少所需存储的数据量。深度压缩算法如算法2所示。

使用深度压缩算法后，在IoT系统中查找异常值的过程如算法3所示。

使用深度压缩算法能在降低数据存储量的同时，满足在IoT系统中查询异常值的要求；以4 KB为单位的NVM读写方式，能发挥NVM在内核态大粒度读写性能较高的优势。

5 写带宽保证的动态调整算法

相比于DRAM，NVM的读写速度不均衡问题更严重。NVM写带宽仅有读带宽的1/3左右，同时DRAM和NVM需要共享内存通道，存在带宽竞争的问题，特别是在多个NVM写操作时，单个写操作的带宽最多会降低54%[13]；新接收的IoT时序数据的存储和历史IoT时序数据的深度压缩均包含NVM写操作。因此，如何合理调节，保证IoT时序数据存储效率是一个重要问题。

首先对历史IoT时序数据的深度压缩进行分解，分析各阶段对新接收的IoT时序数据的存储效率的影响；然后设计多层次动态调整策略，在实现深度压缩的同时，保证IoT时序数据的存储效率。

5.1 IoT时序数据深度压缩的分解

根据历史IoT时序数据深度压缩的过

算法3：IoT系统异常值查找算法

Input：IotTimestamp，dcmp_time，target_block

Output：value

```
1 if IotTimestamp < dcmp_time then//待查数据在上一次深度压缩时间之前
2   target_block = outlierQuery(IotTimestamp, d_blockAddr);//查询深度压缩区d_blockAddr，获取目标数据所在深度压缩块的地址
3   flag = getBlockEflag(target_block);//获取目标深度压缩块的e_flag
4   if flag==1 then//根据目标深度压缩块的构造类型查询数据
5     value = searchDiscreteBlock(IotTimestamp, target_block);//以离散型深度压缩块的构建方式查询目标块，获取目标值
6   else
7     value = searchContinuousBlock(IotTimestamp, target_block);//以连续型深度压缩块的构建方式查询目标块，获取目标值
8   endif
9   if 深度压缩区查询不到待查数据时间戳IotTimestamp then
10    value = NORMAL_VAL;//将目标值设置为正常值
11  endif
12 else//待查数据的时间戳在上一次深度压缩之后
13  value = blkQuery(IotTimestamp,l_blockAddr);//直接在轻量级压缩区中进行查询，获取目标值
14 endif
15 return value
```

程和所涉及的存储设备，将其分解为3个阶段。第一是轻量级压缩块读阶段，该阶段读取存储在NVM中的轻量级压缩块，会产生NVM的读操作。第二是深度压缩缓冲区准备阶段，该阶段将包含异常IoT时序数据的轻量级压缩块写入DRAM中的深度压缩缓冲区，为后续的深度压缩做准备，需要消耗DRAM写带宽。第三是深度压缩块写回阶段，该阶段将构建好的深度压缩块写入NVM中，需要消耗NVM的写带宽。

根据文献[3]可知，NVM读和DRAM读写操作对NVM写带宽的影响较小，但多个NVM写操作之间的互相干扰很大。因此，历史IoT时序数据深度压缩的前两个阶段对IoT时序数据的存储效率的影响较小，而深度压缩块写回阶段则会对IoT时序数据的存储效率造成严重影响。本文采用多层次动态的方法，对IoT时序数据深度压缩中的各阶段压缩分别进行调节，保证IoT时序数据的存储效率。

NVM的读带宽和DRAM写带宽都可以通过测量或计算得到，文献[3]给出了NVM写带宽$NVWB_T$的测量方法，但无法区分不同应用所产生的NVM写带宽。为此，本文首先对深度压缩写操作进行计数，并与内核计时器配合，根据深度压缩的累计写入次数以及操作的总时间，计算深度压缩的NVM写带宽$NVWB_D$；然后，在用户态的轻量级压缩算法中，使用并修改PMDK中与NVM写带宽相关的API，获取轻量级压缩块写回的NVM写带宽$NVWB_W$，并通过系统调用将其传输到内核态。$NVWB_T$、$NVWB_D$将作为调节IoT时序数据深度压缩操作的依据。

5.2 多层次动态调整策略

为了调节IoT时序数据深度压缩操作，针对对IoT时序数据存储效率影响最大的深度压缩块写回操作，在DRAM中构建一个256 MB的深度压缩块写回缓冲区DcmpBuffer，用于暂存已构建的深度压缩块。此外，设置dcmp_threshold1为IoT时序数据深度压缩操作的下限调节阈值，设置dcmp_threshold2为上限调节阈值。

在获取NVM的读带宽、DRAM的写带宽、NVM总写带宽$NVWB_T$、深度压缩块写回NVM的带宽$NVWB_D$和新轻量级压缩块写入NVM的带宽$NVWB_W$的基础上，针对以下4种情况设计深度压缩操作的多层次动态调整策略。

情况1：当$NVWB_T$满足式（2）时，表明当前使用的NVM写带宽较低。此时，不对IoT时序数据深度压缩操作做任何限制，构建的深度压缩块跳过DcmpBuffer直接写回NVM中。

$$NVWB_T < dcmp\_threshold1 \quad (2)$$

情况2：当$NVWB_T$、$NVWB_D$与$NVWB_W$满足式（3）时，表明深度压缩块写回操作已经占用较多的NVM写带宽。因此，将新产生的深度压缩块暂存在DcmpBuffer中，同时通过Intel MBA（Intel memory bandwidth allocation）的可编程请求速率控制器，增加深度压缩块写回阶段中写NVM操作的延迟，限制深度压缩所消耗的NVM写带宽，保证新接收的IoT时序数据的存储效率。

$$dcmp\_threshold1 \leqslant NVWB_T < dcmp\_threshold2,\quad NVWB_D < 0.5 \times NVWB_W \quad (3)$$

情况3：当$NVWB_T$、$NVWB_D$与$NVWB_W$满足式（4）时，表明深度压缩块写回操作占用的NVM写带宽过高。因此，暂停深度压缩块写回阶段的所有操作，在DcmpBuffer还有空闲空间时，缓存所有新产生的深度压缩块，优先存储新接收的IoT时序数据。

$$\begin{aligned} &dcmp\_threshold1 \leqslant NVWB_T < \\ &dcmp\_threshold2, \\ &NVWB_D \geqslant 0.5 \times NVWB_w \end{aligned} \quad (4)$$

情况4：当$NVWB_T$、$NVWB_D$与$NVWB_W$满足式(5)时，表明NVM的写带宽已接近饱和，继续执行深度压缩操作会对新接收的IoT时序数据的存储效率产生较大影响。因此，暂停深度压缩所有阶段的操作，最大限度保证新接收的IoT时序数据的写带宽。

$$\begin{aligned} &NVWB_T \geqslant dcmp\_threshold2, \\ &NVWB_D \geqslant 0.5 \times NVWB_w \end{aligned} \quad (5)$$

使用写带宽保证的动态调整算法，可以缓解深度压缩IoT时序数据与存储新接收的IoT时序数据之间对NVM写带宽的竞争。在优先保证新接收的IoT时序数据的存储效率的情况下，通过多层次的细粒度调整策略，尽可能降低对深度压缩IoT时序数据的影响。

6 原型与测试

为了验证面向NVM的IoT时序数据多态协作压缩策略的性能，笔者在用户态的PMDK中实现轻量级压缩算法，并获取$NVWB_W$；在英特尔傲腾持久内存的驱动程序PMEM中，增加深度压缩与写带宽保证动态调整模块；新增相应系统调用于跳过文件系统，减少冗长I/O软件栈对读写NVM效率的影响，从而实现了面向NVM的IoT时序数据多态协作压缩策略存储系统的原型PCCTSMS。

在测试中，将PCCTSMS中IoT_C的max_val和min_val分别设置为0和9 000，深度压缩间隔dcmp_interval设置为5 min，dcmp_threshold1设置为NVM最大写带宽的10%，dcmp_threshold2设置为NVM最大写带宽的50%。

为了评估原型系统的性能，本文基于NVM构建了3种被广泛使用的时序数据存储系统的原型与PCCTSMS进行比较，分别是InfluxDB1.8.2、OpenTSDB2.4.0（基于HBASE2.1.0）和KairosDB1.2.0（基于Apache Cassandra），这3个原型系统均以EXT4为文件系统。为了验证写带宽保证的动态调整算法的有效性，本文构建了一个移除了动态调整模块的PCCTSMS版本，称之为PCCTSMS w/o DAA。原型系统运行的软硬件环境见**表1**。

真实IoT系统中的数据由于保密等原因难以获取，且部分公开的数据也难以充分体现IoT时序数据的特性。Oliver Kopp等人在存储系统专用测试工具YCSB的基础上，实现了IoT时序数据存储系统的专用测试工具YCSB-TS，本文将其作为测试原型系统的工具。首先，改变IoT时序数据存储系统的数据量，使用**表2**所示的两个工作负载，测试4个原型系统的写、随机查询和范围查询操作的吞吐率；测试时，原型系统的工作线程设置为8个，产生的时序数据值范围设置为0~10 000，其他参数设置为缺省值。其次，测试各原型系统在多线程下的并发性能，同样使用**表2**所示的工作负载，分别设置工作线程数为1、4、8、16和32，在数据量为100万条的情况下，测试各原型系统的吞吐率。此外，针对原型系统中不同比例的异常IoT时序数据，测试各原型系统的数据压缩率。

6.1 写吞吐率的测试

使用YCSB-TS执行load阶段，并测试各个原型系统在写入不同数据量的情况下的写吞吐率，结果如**图5**所示。

由**图5**可知，PCCTSMS的写吞吐率均

高于其他原型系统，PCCTSMS的写吞吐率相比InfluxDB提高了22.7% ~ 33.8%，相比OpenTSDB最高提升了161.4%，相比KairosDB提高了42.9% ~ 71%。这是因为PCCTSMS设计的轻量级压缩算法在用户态对接收的IoT时序数据进行了快速压缩，并在构建轻量级压缩块后才写入NVM中，显著减少了写入IoT时序数据的数据量；同时，所设计的256 B轻量级压缩块能很好地适应用户态写NVM的特性，保持良好的写入速度。随着在原型系统中写入的数据量从100万条增加到2 000万条，InfluxDB、OpenTSDB与KairosDB的写吞吐率分别下降了22.9%、50.4%和29.4%，而PCCTSMS仅仅下降了16%，这表明PCCTSMS的写性能相比其他原型系统具有更强的稳定性。OpenTSDB底层使用HBase存储数据，I/O软件栈过长，难以发挥NVM的读写优势，同时OpenTSDB还需要维护大量的索引，增加了写入操作的复杂度和额外开销，导致在存储大量数据时写吞吐率严重下降。InfluxDB与KairosDB采用LSM-tree作为存储结构，存在额外的合并开销和写入放大等问题，影响了存储大量IoT时序数据的写入效率。而PCCTSMS利用IoT时序数据的特性，避免了维护复杂索引等额外开销。

此外，在YCSB-TS中load阶段将并发工作线程数分别设置为1、4、8、16和32，测试不同原型系统的写吞吐率，结果如图6所示。

由图6可知，PCCTSMS的写吞吐率均高于其他原型系统，相比于InfluxDB、OpenTSDB和KairosDB分别提升了21.1% ~ 22.4%、50.8% ~ 68.9%和9.4% ~ 41.5%。随着并发工作线程数的增加，所有原型系统的写吞吐率不断上升，并在8线程时达到峰值，随后各原型系统的写吞吐率均出现下降。在写线程增加到32时，相比各原型系统最高的写吞吐率，InfluxDB下降了20.6%，OpenTSDB下降了24.6%，

表1 原型系统的测试环境

部件	配置
CPU	Intel(R) Xeon(R) Platinum 8222L 3.00 GHz
Memory	128 GB
NVM device	2 × 128 GB Intel Optane DC Persistent Memory
OS	CentOS 7.0, Kernel 5.18.18

表2 测试的两个负载

负载	配置
tsworkloadA	100% read
tsworkloadB	100% scan

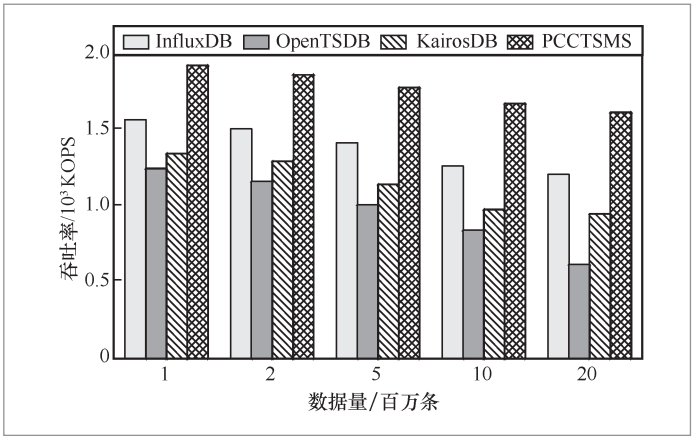

图5 改变数据量的写吞吐率

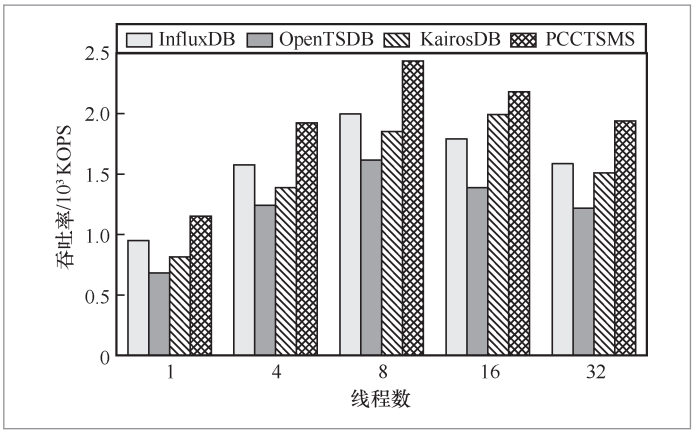

图6 改变工作线程数的写吞吐率

KairosDB下降了18.5%，PCCTSMS下降了20.3%。这是因为CPU的64字节访问大小与NVM的256字节访问大小不匹配，在高并发环境下会引发NVM内部的缓存抖动问题[12]；OpenTSDB底层的HBase需要运行在Java虚拟机上，其垃圾回收等机制会影响其运行性能或导致写入操作被暂停。

6.2 随机查询吞吐率的测试

使用YCSB-TS的tsworkloadA负载，通过改变原型系统中的IoT时序数据量，测试各原型系统的随机查询吞吐率，测试结果如**图7**所示。

由**图7**可知，PCCTSMS的随机查询吞吐率均高于其他原型系统，PCCTSMS相比InfluxDB提高了3.2% ~ 12.1%，相比OpenTSDB最大提高了113.2%，相比KairosDB提高了17.9% ~ 31.2%。因为轻量级压缩算法和深度压缩算法能减少存储的IoT时序数据量，同时深度压缩块集中存储了异常的IoT时序数据，能提高查询效率。KairosDB使用Google的Snappy压缩算法，其具有较高的压缩和解压速度，但未利用IoT时序数据的特点来提高数据的压缩率，此外Snappy算法还会破坏数据的局部性，影响访问性能。PCCTSMS能利用IoT时序数据查询的时间特性，判断其位于深度压缩区还是轻量级压缩区，缩小了查询范围，提高了查询效率。

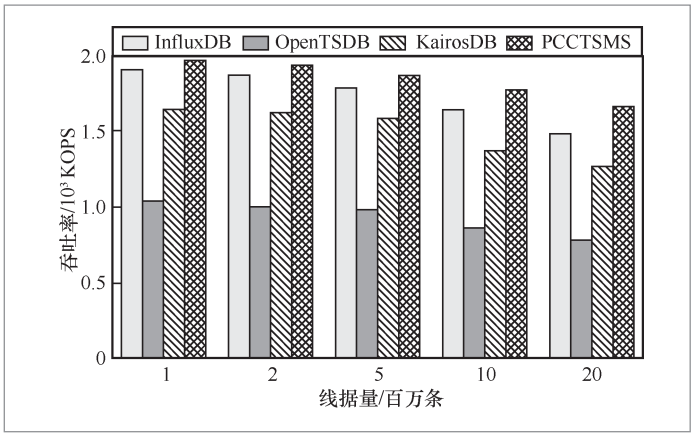

图 7 改变数据量的随机查询吞吐率

此外，在YCSB-TS中使用tsworkloadA，将并发工作线程数分别设置为1、4、8、16和32，测试各个原型系统的并发随机查询吞吐率，结果如**图8**所示。

图8的测试结果表明，在多线程环境下，PCCTSMS的随机查询吞吐率与InfluxDB、OpenTSDB和KairosDB相比，分别提升了1.7% ~ 3.7%、42.7% ~ 70.8%和4.3% ~ 26.6%，这同样是因为NVM内部的缓存抖动问题。所有原型系统都在工作线程数为16时达到并发随机查询吞吐率的峰值，随后出现下降。随着工作线程数的增加，KairosDB与InfluxDB的随机查询吞吐率差距不断缩小，由24.1%下降为1.3%。InfluxDB使用时序数据缓存提高查询效率，但增加工作线程数会影响缓存的命中率，从而降低InfluxDB的随机查询吞吐率；而KairosDB采用的数据分片方法和线程池等方法，能更好地支持多个工作线程的并发访问。此外，InfluxDB与KairosDB都采用LSM-tree作为存储结构，但LSM-tree支持并发访问的能力一般，同时合并操作也会进一步影响并发访问性能。PCCTSMS中的轻量级压缩块大小能适应NVM的读写特性，同时IoT时序数据管理结构简单，能较好地支持大量并发工作线程的访问，保证了多线程环境下的随机查询吞吐率。

6.3 范围查询吞吐率的测试

使用YCSB-TS中tsworkloadB工作负载，改变查询的数据量，测试各原型系统范围查询的吞吐率，结果如**图9**所示。

由**图9**可知，PCCTSMS的范围查询吞吐率同样高于其他原型系统，相比于

InfluxDB、OpenTSDB和KairosDB分别提升了3.6% ~ 10.5%、71.2% ~ 172.8%和17.1% ~ 26.9%。随着数据量从100万增加到2 000万，InfluxDB、OpenTSDB、KairosDB与PCCTSMS的范围查询吞吐率分别下降了18.3%、45.4%、19.8%和13.1%。因为InfluxDB以TSM为索引结构，通过时间范围分区提高查询效率；OpenTSDB扫描HBase中行键和列族的时间开销大；KairosDB可以利用Cassandra的数据分片策略提高查询效率；PCCTSMS不需要维护复杂的索引结构，并能通过减少存储的IoT时序数据量和依据时序构建数据块等方式在提高范围查询吞吐率的同时，保持查询吞吐率的稳定。

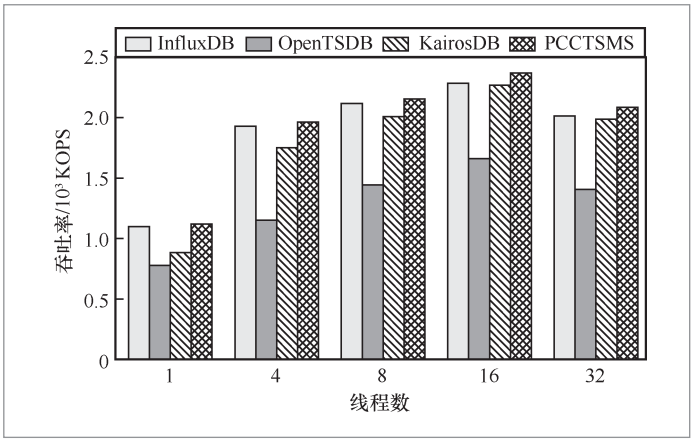

图 8 改变工作线程数的随机查询吞吐率

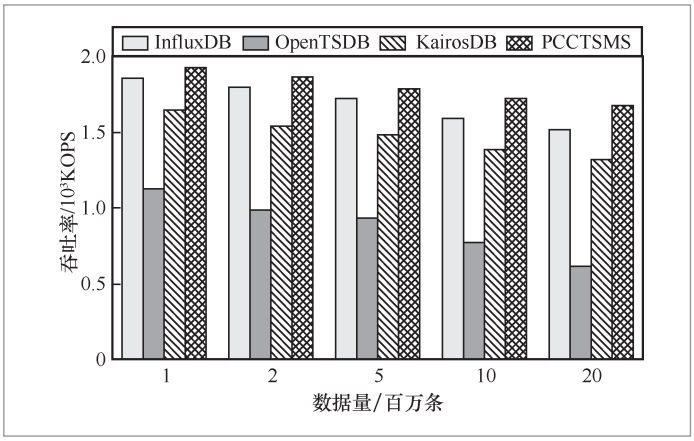

图 9 改变数据量的范围查询吞吐率

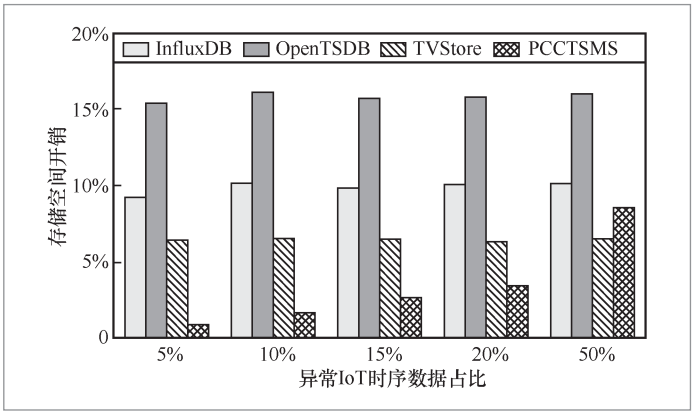

图 10 不同比例异常数据量所占用的存储空间

6.4 IoT时序数据压缩率的测试

构建5组包含200万条数据的IoT时序数据集，其中IoT时序数据的范围为0 ~ 10 000，异常值的比例依次为5%、10%、15%、20%和50%。将这些数据集分别存储到各个原型系统中，测试其所占用的存储空间，获得各原型系统对IoT时序数据压缩率。PCCTSMS中将min_val设置为0，max_val分别为9 500、9 000、8 500、8 000、5 000，其他原型系统均采用缺省设置，测试结果如**图10**所示。

由**图10**可知，随着异常IoT时序数据占比的增加，InfluxDB、OpenTSDB和TVStore的存储空间开销并未发生明显波动。在异常IoT时序数据占比到达50%之前，PCCTSMS所需的存储空间最小，相比于InfluxDB、OpenTSDB和TVStore分别少占用6.6% ~ 8.5%、12.4% ~ 14.6%和2.9% ~ 5.5%。因为PCCTSMS所设计的深度压缩算法能缩小占用的存储空间，而其他的原型系统缺乏针对IoT系统以查询异常IoT时序数据为主的特性的优化策略。随着异常IoT时序数据值占比达到50%，PCCTSMS所需存储空间仍然低于InfluxDB和OpenTSDB，但高于

TVStore。TVStore比PCCTSMS节省了2%的存储空间，因为TVStore采用的时变压缩策略仅使用产生时间判断数据的重要性。此外，TVStore还限定了最大存储空间，但其存在丢失历史异常IoT时序数据的问题，难以满足IoT系统的要求。

6.5 消融实验

用户态轻量级压缩算法与深度压缩算法相互关联，共同完成IoT时序数据的压缩任务。本节将PCCTSMS中写带宽保证的动态调整模块去除，实现了PCCTSMS w/o DAA，将其与PCCTSMS进行对比测试。使用YCSB-TS执行load阶段，并测试两个原型系统在写入不同数据量情况下的写吞吐率，结果如**图11**所示。

图11的结果表明，PCCTSMS相比PCCTSMS w/o DAA具有更高的写吞吐率。随着写入数据量从100万增加到2 000万，相较于PCCTSMS w/o DAA，PCCTSMS的写吞吐率提升幅度从21.4%增长到41.4%。这一结果表明深度压缩对NVM写带宽的影响随数据规模的扩大而增加，同时也验证了写带宽保证的动态调整算法能够缓解轻量级压缩块的深度压缩操作与存储新接收的IoT时序数据操作对NVM写带宽的竞争，保证了新接收的IoT时序数据的存储效率。

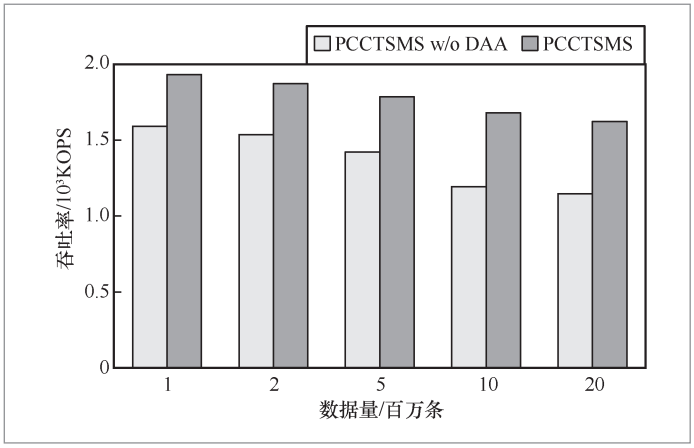

图 11 写带宽保证的动态调整算法对写吞吐率的影响

7 结束语

随着IoT系统的快速发展和应用，IoT时序数据的存储与管理是一个急需解决的重要问题。本文提出了面向NVM的IoT时序数据多态协作压缩策略，针对NVM与IoT时序数据的特性，进行分层压缩，在减少需存储的IoT时序数据的数据量的同时，保证了IoT时序数据的存储效率，保证了IoT系统中对异常时序数据的长期查询和分析。实验表明，与InfluxDB、OpenTSDB、KairosDB以及TVStore相比，本文提出的方法最高能提升161.3%的写吞吐率，减少14.6%的存储空间开销。

参考文献:

[1] 陈游旻，李飞，舒继武. 大数据环境下的存储系统构建：挑战、方法和趋势[J]. 大数据，2019, 5(4): 27-40.
CHEN Y M, LI F, SHU J W. Building storage systems in big data era: challenges, methods and trends[J]. Big Data Research, 2019, 5(4): 27-40.

[2] 王杰. 多态协作的高并发NVM存储系统[D]. 镇江：江苏大学，2020.
WANG J. The kernel and user space collaborative and highly concurrent NVM storage system[D]. Zhenjiang: Jiangsu University, 2020.

[3] YI J F, DONG B C, DONG M K, et al. MT²: memory bandwidth regulation on hybrid NVM/DRAM platforms[C]//Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST 22). Berkeley: USENIX Association, 2022: 199-216.

[4] AN Y Z, SU Y, ZHU Y Q, et al. TVStore:

automatically bounding time series storage via time-varying compression[C]//Proceedings of the 20th USENIX Conference on File and Storage Technologies(FAST 22). Berkeley: USENIX Association, 2022: 83-100.

[5] BLALOCK D, MADDEN S, GUTTAG J. Sprintz: time series compression for the Internet of Things[J]. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2018, 2(3): 1-23.

[6] PAUL D, PENG Y Q, LI F F. Bursty event detection throughout histories[C]// Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE). Piscataway: IEEE Press, 2019: 1370-1381.

[7] KHELIFATI A, KHAYATI M, CUDRÉ-MAUROUX P. CORAD: correlation-Aware Compression of Massive Time Series using Sparse Dictionary Coding[C]//Proceedings of the 2019 IEEE International Conference on Big Data (Big Data). Piscataway: IEEE Press, 2019: 2289-2298.

[8] CAI T, LIU P Y, NIU D J, et al. The embedded IoT time series database for hybrid solid-state storage system[J]. Scientific Programming, 2021, 2021: 9948533.

[9] HAN S K, JIANG D J, XIONG J. SplitKV: splitting IO paths for different sized key-value items with advanced storage devices[C]//Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20). Berkeley: USENIX Association, 2020: 1-18.

[10] KAIYRAKHMET O, LEE S Y, NAM B, et al. SLM-DB: single-level key-value store with persistent memory[C]//Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST 19). Berkeley: USENIX Association, 2019: 191-205.

[11] CHEN Y M, LU Y Y, ZHU B H, et al. Scalable persistent memory file system with kernel-userspace collaboration[C]//Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21). Berkeley: USENIX Association, 2021: 81-95.

[12] ZHOU D Y, QIAN Y C, GUPTA V, et al. ODINFS: scaling PM performance with opportunistic delegation[C]//Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). Berkeley: USENIX Association, 2022: 179-193.

[13] MA S N, CHEN K, CHEN S M, et al. ROART: range-query optimized persistent ART[C]//Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21). Berkeley: USENIX Association, 2021: 1-16.

作者简介

蔡涛(1976-),男,博士,江苏大学计算机科学与通信工程学院副教授,主要研究方向为网络存储系统与NVM。

雷天乐(1999-),男,江苏大学计算机科学与通信工程学院硕士生,主要研究方向为存储系统与NVM。

牛德姣(1978-),女,博士,江苏大学计算机科学与通信工程学院副教授,主要研究方向为存储系统与神经网络。

戴健飞(1999-),男,江苏大学计算机科学与通信工程学院硕士生,主要研究方向为存储系统与NVM。

黄泽宇(1998-),男,江苏大学计算机科学与通信工程学院硕士生,主要研究方向为存储系统与NVM。

倪强强(1998-),男,江苏大学计算机科学与通信工程学院硕士生,主要研究方向为存储系统与NVM。

收稿日期: 2024-04-15
通信作者: 蔡涛, caitao@ujs.edu.cn
基金项目: 国家重点研发计划项目(No.2019YFB1600500)
Foundation Item: The National Key Research and Development Program of China (NO.2019YFB1600500)