小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems
DOI:10.20009/j.cnki.21-1106/TP.2023-0586
2025 年 4 月 第 4 期
Vol.46 No.4 2025

深入理解 IoTDB 在时序数据工作负载下的性能表现

刘 健,程建勋,闫高锋,吕泽涛,孙国道,梁荣华,蒋 莉

(浙江工业大学 计算机科学与技术学院,杭州 310023)
E-mail:jliu83@zjut.edu.cn

摘 要: IoT 设备的大量部署导致时序数据呈现爆炸式增长,为了更好地管理大规模时序数据,高效满足上层时序用户的高性能访问需求,各种时序数据库被广泛开发和部署.但目前从业人员对于时序数据库的理解还不够深入,积累的知识和经验往往局限于传统数据库,并不适用于新兴的时序数据库.本文以面向 IoT 设计的开源时序数据库 IoTDB 为基础,利用基准测试工具 IoT-Benchmark 生成多样化的时序数据工作负载,深入探究 IoTDB 在处理时序数据工作负载时的性能表现和行为特征.本文的研究能够为时序数据库的设计者和开发者在开发和部署高性能时序数据库方面提供更有价值的帮助和指导.

关 键 词: IoTDB;时序数据工作负载;性能评估;新硬件
中图分类号: TP392 **文献标识码:** A **文 章 编 号:** 1000-1220(2025)04-1014-11

In-depth Understanding on Performance of IoTDB in Handling Time-series Data Workloads

LIU Jian, CHENG Jianxun, YAN Gaofeng, LÜ Zetao, SUN Guodao, LIANG Ronghua, JIANG Li
(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract: Time-series data is growing explosively driven by the rapid proliferation of IoT devices. To more efficiently manage the large-scale time-series data and meet the high-performance access demands of upper-layer time-series users, various time-series databases have been developed and deployed. Despite these efforts, practitioners currently lack a thorough understanding of time-series databases remains insufficient, with accumulated knowledge and experience predominantly confined to traditional databases. This limitation hinders the seamless adaptation of practices to emerging time-series databases. This paper focuses on addressing these challenges by leveraging the open-source time-series database, IoTDB, specifically designed for IoT applications. Employing the benchmarking tool IoT-Benchmark, diverse time-series data workloads can be generated for the comprehensive evaluation on the performance of IoTDB. The objective is to conduct a thorough investigation into the performance behaviors of IoTDB in handling time-series data workloads. The research in this paper can provide insightful guidance and help for system designers and developers to develop and deploy high-performance time-series databases.

Keywords: IoTDB; time-series data workloads; performance evaluation; emerging devices

0 引 言

物联网(Internet of Things,IoT)技术的快速发展促使 IoT 设备在各个领域大量部署,如环境监测[1]、医疗保健[2]、自动驾驶[3]等,从而导致时序数据呈现爆炸式增长.例如,一个仅配有 1000 个电表的小型电网每秒钟可以产生 140 万个数据点[4].为了更高效地处理快速增长的时序数据,时序数据库得到了广泛的开发和部署,并引起了学术界和工业界的普遍关注[3-11].

与传统数据库(如关系型数据库和非关系型数据库)不同,时序数据库是专为应对时序数据工作负载而量身定制的系统解决方案,其主要特点包括以下 3 个方面:1)时间范围导向的查询通常在时序数据库操作中占主导地位[12,13].时序数据通常用于周期性监测、行为分析、或趋势预测,这些分析通常基于一系列随时间变化的数据点,而对单个数据点的关注度相对较低.因此,时序数据库必须能够有效地支持时间范围导向的查询操作,如在指定的时间窗口内对大量数据点进行聚合计算;2)时序数据库必须高效地处理乱序数据[7,9].时序数据库通常采用时间序列数据模型,这为时间范围导向的查询奠定了基础,同时也要求时序数据必须按照正确的时间顺序进行存储.然而,由于数据源的多样性、传输延迟等因素,时序数据的写入并不能保证始终遵循严格的时间顺序.因此,必须采取有效的策略来处理乱序数据,以提高存储效率和查询性能,如定期在后台运行数据整合操作;3)数据压缩已成为时序数据库中不可或缺的重要组成部分[6-9].随着 IoT 设备的不断增多,采样频率的不断提高,采样过程的高度自动化,时

收稿日期:2023-12-07 收修改稿日期:2024-02-05 基金项目:国家自然科学基金项目(62202430)资助;浙江省自然科学基金杰出青年科学基金项目(LR23F020003)资助;浙江省基础公益研究计划项目(LTGG23F020005)资助. 作者简介:刘 健,男,1988 年生,博士,讲师,CCF 会员,研究方向为数据库系统、存储系统;程建勋,男,1999 年生,硕士,研究方向为时序数据库;闫高锋,男,1991 年生,硕士,研究方向为信息可视化与可视分析;吕泽涛,男,2000 年生,硕士,研究方向为存储系统;孙国道,男,1988 年生,博士,教授,CCF 会员,研究方向为信息可视化与可视分析;梁荣华,男,1974 年生,博士,教授,博士生导师,CCF 高级会员,研究方向为可视化和计算机图形图像处理;蒋 莉(通信作者),女,1976 年生,博士,副教授,研究方向为信息可视化与可视分析.

序数据规模持续快速扩大. 由于时序数据自身固有的高冗余特性,时序数据压缩已成为缓解时序数据存储压力的重要手段. 例如,Facebook 提出了一种针对时序数据特点高度优化的压缩编码 Gorilla[14],已被广泛应用于主流时序数据库中[7,14].

然而,目前从业人员对于数据库的理解往往局限于传统数据库范畴,上述差异意味着建立在传统数据库上的先验知识可能已不再适用于新兴的时序数据库. 并且,在对时序数据库的深入理解方面,尤其是在处理独特的时序数据工作负载时,学术界对时序数据库的理解还不够清晰. 为了填补这一知识鸿沟,本文计划以面向 IoT 设计的列式时序数据库 IoTDB[7](Database for Internet of Things)为基础,利用测试基准工具 IoT-Benchmark[15]生成多样化的时序数据工作负载,开展一系列全面、深入和详实的实验研究,从而获得更多关于时序数据库特定行为的见解. 具体来说,本文希望回答以下 4 个重要的科学问题:

1)不同的时序数据工作负载模式(例如,数据类型、乱序数据分布、时间范围长度等)可能会对 IoTDB 的行为产生多方面的影响,这些不同的行为对于 IoTDB 性能又有什么样的影响?

2)IoTDB 会周期性地在后台运行数据整合操作来处理乱序数据,而后台的数据整合操作可能会与前端的数据访问发生资源竞争(如 CPU、I/O),那么数据整合操作对于 IoTDB 的性能有什么样的影响呢?

3)时序数据压缩模块作为 IoTDB 的重要组成部分,涵盖了多种不同的编码方式和压缩算法,例如 TS_2DIFF、Gorilla、LZ4 等. 尽管数据压缩可以有效减少存储空间的消耗,但这也伴随着进一步增加的 CPU 资源使用. 那么不同的编码和压缩算法对 IoTDB 的性能有什么影响?

4)IoTDB 具有 I/O 密集型和数据密集型的特点,因此,底层存储设备对于 IoTDB 的性能有着决定性的影响. 本研究选取了 3 种具有代表性的存储设备:叠瓦式磁盘(Shingled Magnetic Recording, SMR)、传统的机械硬盘(Hard Disk Drive,HDD)、以及新兴的固态硬盘(Solid State Drive,SSD). 这些存储设备在存储密度、读写性能方面呈现显著的差异,那么它们对于 IoTDB 性能的影响如何?

通过回答上述问题,本文将深入探讨在应对时序数据工作负载时与时序数据库相关的性能行为,这将有助于提高相关研究人员对该领域的深刻理解,并期望本文的研究结果和深入分析能够为时序数据库系统设计师和从业者提供更有价值的帮助,从而推动和发展更为高效的、切实可行的时序数据库系统解决方案.

1　背景及相关工作

Apache IoTDB 由清华大学研发,专为面向 IoT 应用而设计的列式开源时序数据管理系统,其轻巧的架构设计和出色的读写性能,使其成为广泛适用于多元化物联网场景的时序数据库. IoTDB 的设计基于面向写性能优化的数据结构日志合并树(Log-Structured Merge Tree,LSM-Tree),以高效应对时序数据的高并发写入,其核心理念在于将小粒度、随机的数据写入转化为大规模、顺序的、批量化的数据写入操作. 为了进一步提升写入性能,所有新写入数据点会以 Memtable 的形式缓存于内存中,且系统维护两类 Memtables:一类用于管理按顺序到达的数据,另一类用于处理乱序到达的数据. IoTDB 通过维护一个时间探测器(Time Detector)来判断数据是否为顺序到达. 该探测器为每个设备记录已写入磁盘中数据的最大时间戳. 当下一个数据点的时间戳大于该时间戳时,系统将其写入正常内存表(Normal Memtable);否则将其写入延迟内存表(Delayed Memtable)中. 在满足一定的阈值条件时,Memtables 被分别刷写到更低级存储设备,形成两类 TsFiles,即普通 TsFile 和延迟 TsFile. 鉴于 TsFile 中的数据点可能在时间范围上存在重叠,IoTDB 定期在后台运行数据整合操作,将它们合并成相互不重叠的普通 TsFiles,以提高数据查询效率. 图 1 为 IoTDB 的 LSM-Tree 结构示意图.

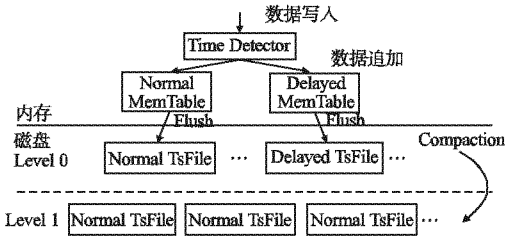

图 1　IoTDB 中的 LSM-Tree 结构示意图

Fig. 1　Structure diagram of LSM-Tree in IoTDB

近年来,时序数据库的研究引起了学术界和工业界的广泛关注,但大多数先前的研究侧重于设计和优化时序数据库,以满足高性能时间序列数据访问的需求. 例如,InfluxDB 和 IoTDB 通常采用类似 LSM-Tree 的数据结构,以提高数据写入速度[6,7]. 基于内存的时序数据库,如 Monarch[11]、Gorilla[14]和 ByteSeries[16],充分利用内存的高吞吐量和低延迟,显著提高查询效率. 一些基于闪存 SSD 设计的高效时序数据缓存方案,如 TSCache[13],能够显著提升 InfluxDB 的查询性能.

针对时序数据库的测试研究工作相对较少,大多先前研究仍集中在传统数据库领域. 例如,Wang 等人在真实工作负载下对时序数据库 IoTDB 进行了测试,并与其它时序数据库(如 InfluxDB、TimescaleDB、KairosDB)进行了比较,展示了 IoTDB 和 Tsfile 在工业物联网场景中的优越性[17]. 基准测试是一种评估系统、应用程序或组件性能的方法,Liu 等人开发了一个基准测试工具,即 IoT-Benchmark,为时序数据库(如 IoTDB、TimescaleDB)读写性能的评估测试奠定了基础[15]. Hao 等人提出的 TS-Benchmark 基准测试,对 4 个代表性的时序数据库 InfluxDB、TimescaleDB、Druid 和 OpenTSDB 进行了详细比较[18].

而在传统数据库研究方面,Jia 等人基于 RocksDB 进行了大量的实验测试,有针对性地研究了 5 种不同的多目标优化算法,为传统键值数据库的自动调优提供了解决思路[19]. Liu 等人对传统数据库 SQLite、MongoDB、LevelDB 的性能和能耗进行深入研究,实验结果表明合理的参数配置能够显著提升性能并且降低能耗[20]. Raasveldt 等人对数据库管理系统性能比较中的常见陷阱进行了研究,并提供了如何发现和避免它们的建议,以便进行系统间公正的性能比较[21]. Tongkaw

等人对数据库 MySQL 和 MariaDB 的性能进行实验测试比较,实验结果表明,MySQL 的性能明显优于 MariaDB[22]. 王洁等人设计了 TPC-DS_Utensil,通过对 MySQL、Oracle、Hive 和 Hbase 进行性能测试,表明该测试工具能够准确地反映服务器性能[5]. 杜丽娟等人对传统数据库 MySQL 和 NoSQL 数据库 Redis 进行性能测试比对,得出 Redis 数据库的存储和查询效率高的结论[10]. 而 Ray 等人提出的一种空间数据库基准测试工具 Jackpine 是可移植的,可以支持任何具有 JDBC 驱动程序实现的数据库,包括微基准测试和宏工作负载场景[23].

与之前的研究不同,本文主要目标是从系统研究的角度更深入地探究时序数据库在处理时序数据工作负载时的性能表现. 为实现该目标,本文以 IoTDB 为基础,开展广泛的实验研究,覆盖了数据库管理系统的多个方面,包括底层存储设备、内部数据库组件以及上层时序数据工作负载,旨在填补时序数据库在处理时序数据工作负载时性能方面的知识鸿沟,以更全面地理解时序数据库的行为和性能特征.

2 实验方法和配置

2.1 硬件和系统

本文的实验在一台联想 ThinkSystem ST258 服务器上进行,该服务器配备了一颗四核的 Intel Xeon E-2224 3.4GHz 处理器、8GB 内存,并采用了一块容量为 2TB、转速为 7200 RPM (Revolutions Per Minute)的东芝硬盘作为系统盘安装了 64 位的 Ubuntu 18.04 LTS 操作系统,搭载 Linux Kernel 5.4.0 内核,Ext4 文件系统,并且所有实验都是基于 0.13.4 版本的 IoTDB 时序数据库.

表 1 存储设备介绍

Table 1 Introduction of Storage devices

存储设备	写(MB/s)	读(MB/s)	存储容量
三星 SSD	507	547	500GB
西部数据 HDD	227	242	2TB
希捷 SMR	96.6	52.2	2TB

为了深入研究存储设备对数据库性能的影响并最小化对系统盘的干扰,本文引入了额外 3 种不同类型的存储设备,以提供全面的性能评估,包括一块转速为 5400 RPM、容量为 2TB 的希捷 SMR 硬盘,以其高存储密度和低成本著称,但在随机写性能方面可能表现不足;一块转速为 7200 RPM、容量为 2TB 的西部数据 HDD,相比于 SMR 在性能方面有所提升,但仍和 SSD 有一定的差距;一块容量为 500GB 的三星 870 SATA SSD,尽管价格相对较高,但是低延迟和高带宽的卓越性能使其拥有广泛的应用场景. 这 3 种设备在存储密度、成本以及性能方面存在显著差异,为本文的测试研究提供了丰富的硬件支持. 表 1 展示了 3 种不同存储设备的读写性能、存储容量以及相应的价格. 其中,读写带宽是在 Linux 环境下利用 dd 命令在 4KB 的块大小条件下测试得出,且读写数据总量都为 12GB.

2.2 测试基准和工作负载

IoT-Benchmark 是一款专注于评估物联网场景下的时序数据库性能的基准测试工具. 通过调整配置文件,该工具可以生成多样化的时序数据工作负载,以综合评估时序数据库 IoTDB 的性能表现.

在数据生成方面,Device_Number 指定设备的总数,而 Sensor_Number 指定了每个设备连接的传感器数量,这些传感器能够以规则的时间间隔生成不同类型的数据点(包括整数、浮点数等). 为确保数据的一致性并简化数据插入过程,所有数据点都被组织成数据行,每个数据行包含一个设备的所有传感器生成的数据点.

在数据写入方面,基本的写入单元是一个“批次(Batch)”,其中 Batch_Size_Per_Write 参数指定了单个写操作中数据的行数. 为了将连续生成的数据点有效地插入数据库,需要迭代执行一系列的写操作,循环的次数由 Loop 指定. Client_Number 确定进行写入的客户端数量,由这些客户端共同完成对应循环次数的写入任务,总写入操作数为 loop. 为了进一步提高存储效率,IoTDB 已经集成了多种数据压缩技术并将整个数据压缩过程分为两部分,先是将数据行编码转换为二进制流,然后再进行压缩,采用的编码方式如 TS_DIFF、Gorilla,使用的压缩算法如 LZ4 和 SNAPPY,根据特定的数据类型在 IoT-Benchmark 的配置文件中进行配置. 此外,IoT-Benchmark 可以按照 Poisson 分布模拟乱序数据写入模式,乱序数据比例由 Out_Of_Order_Ratio 确定.

对于数据查询,本文选择了以时间范围为导向的聚合查询(Qa)为代表:该查询涉及在指定时间范围内基于特定聚合函数(如 Count 统计该时间范围内的点数)对数据进行聚合计算,同时对设备和数值进行过滤,查询语句为 select func(v1)… from data where device in? and value >? and time >? and time <?.

表 2 查询涉及参数以及说明

Table 2 Parameters involved in the query and their descriptions

查询涉及参数	对应参数说明
Query_Sensor_Num	规定每条查询语句涉及到传感器数量
Query_Device_Num	规定每条查询语句涉及到设备数量
Query_Interval	规定查询语句的时间过滤的起止时间的差值
Step_Size	规定相邻查询语句的起始时间的变化步长
Loop	规定每个客户端进行查询的次数
Client_Number	规定同时有多少个客户端在进行查询操作
Point_Step	规定两个数据点之间的时间间隔

根据表 2 可知,同一客户端相邻两条查询语句时间过滤条件的起始时间相差时长为 Step_Size * Point_Step. 在对应工作负载下,查询操作执行总次数为 Client_Number * Loop.

3 实验结果与分析

为了更好地回答之前提出问题,本文的实验设计主要围绕以下 3 个方面展开:1)评估时序数据工作负载对 IoTDB 性能的影响;2)研究数据库 IoTDB 组件对性能的影响;3)分析底层存储设备对 IoTDB 性能的影响. 在此基础上,借助热力图可视化工具,直观地展示数据库多性能指标之间以及性能

指标与参数之间的关系,为更好地理解和分析 IoTDB 的性能提供了有益的支持. 鉴于参数组合的多样性,为深入研究,本文选取了代表性的关键参数和模块进行实验.

默认参数配置:对于数据写入,采用顺序的写入方式,且数据类型设置为 DOUBLE 类型,为了更好地反映数据 I/O 的真实情况,将数据的编码方式设置为 PLAIN(不对数据进行编码),压缩算法的设置为 UNCOMPRESSED(不对数据使用压缩算法),Client_Number 设为 16,Point_Step 设为 400ms,Batch_Size_Per_Write 为 100,Loop 为 10000;对于数据查询,每条查询语句的 Query_Sensor_Num 设为 10,Query_Device_Num 设为 5,相邻两条查询语句时间间隔 Query_Interval 设为 1800s;为了确保对同一时间序列数据查询的时间不重叠,将 Step_Size 为 4500,对应的 Point_Step 为 400ms,Step_Size 和 Point_Step 的乘积正好是 1800s,Loop 设为 40,Client_Number 设为 16.

在后续的实验过程中,若没有特别说明,使用默认的参数配置. 对于下文出现的双轴图,黑色是写入情况对应左轴;灰色是查询情况对应右轴. 对于下文表格中出现的简写,W 代表是写入,Q 代表的是查询,基准线数据都是具体数值.

3.1　不同时序数据工作负载模式的影响

为探索不同的时序数据工作负载模式对于数据库性能的影响,本文选取了客户端数量(Client Number)、查询的时间范围(Time Interval)、数据类型(Data Type)、乱序数据(Out-of-order Data)4 个关键参数进行测试研究.

表 3 展示了不同的客户端数量对于数据库的读写性能的影响,基准线为客户端数量取 16. 观察结果显示,随着 Client 数量的增加,读写性能在不断提升,当 Client 数为 16 的时候,读写性能基本上达到峰值. 例如,使用 16 个 Client 进行数据读写时,相对于 1 个 Client 的情况,就吞吐量而言,写入性能提升了 21.8%,查询性能提升了 1.2 倍,其主要原因是更多的 Client 并发地去处理工作负载,能够更充分利用系统的资源. 例如,CPU 资源相对于 1 个 Client 提升了 40 个百分点. 但随着 Client 数量继续增加到 64,读写的延迟都增加了 4 倍,同时系统写入吞吐量降低了 19.9%. 这是因为系统的资源利用已经趋于饱和,无法进一步提升性能. 另外,可以发现查询的 I/O 数据总量随着 Client 的数目增多也在增加,这是因为查询操作的执行数量为 Client_Number 和 Loop 的乘积,客户端数量越大,进行的查询操作就越多,对底层数据的读取量就越大.

表 3　客户端数量对数据库读写性能的影响对比

Table 3　Comparison of the impact of the number of clients on database read and write performance

Client Number	W Throughput (Point/Sec)	W AVG Latency(ms)	W AVG CPU Usage(%)	W I/O Amount(GB)	Q Throughput (Point/Sec)	Q AVG Latency(ms)	Q AVG CPU Usage(%)	Q I/O Amount(GB)
1	21.79% ↓	92.87% ↓	52.08% ↓	16.84% ↓	50.31% ↓	87.93% ↓	63.71% ↓	92.18% ↓
4	7.68% ↓	73.47% ↓	19.93% ↓	2.44% ↑	9.02% ↓	73.31% ↓	0.79% ↓	73.58% ↓
16	10986654.54	27.37	80.92	184.92	140.62	5489.26	52.99	86.93
32	12.12% ↓	106.13% ↑	1.64% ↓	1.26% ↑	1.24% ↑	101.06% ↑	0.51% ↑	99.59% ↑
48	14.91% ↓	219.25% ↑	0.07% ↑	2.43% ↑	1.27% ↑	201.93% ↑	0.88% ↓	200.52% ↑
64	19.98% ↓	309.53% ↑	0.48% ↑	6.12% ↑	0.12% ↓	314.39% ↑	0.73% ↑	303.09% ↑

小结:随着 Client 数量的增加,系统资源得以更加充分地利用,数据库性能显著提升. 然而,过多的客户端数量可能导致系统资源使用过饱和,性能反而受损. 综合实验结果分析,在当前的实验环境下,采用 16 个客户端进行数据库的读写操作可能是一个合适的选择,这配置在性能提升和系统资源充分利用之间取得了平衡.

查询的时间范围对于数据聚合操作(Count)的性能影响明显,随着查询时间范围的增大,进行 Count 统计的数据点也随之增多,从而导致系统的查询性能降低. 表 4 为不同查询时间范围 IoTDB 进行查询时的性能,基准线为 1800s,如其所示,相对于查询时间范围是 600s 的查询请求,当查询时间范围增加到 3000s 时,系统的 I/O 数据总量增加了 5.3%,这也导致了系统吞吐量降低了 12.9%,平均延迟增加了 15.0%.

表 4　不同时间范围的查询性能表

Table 4　Query performance table for different time ranges

Time Interval(s)	Throughput (Point/Sec)	Average Latency(ms)	Average CPU Usage(%)	I/O Amount (GB)
600	4.99% ↑	5.93% ↓	3.62% ↓	0.72% ↓
1200	3.22% ↑	3.67% ↓	56.61	0.57% ↓
1800	195.61	3988.87	55.52	64.04
2400	2.88% ↓	2.78% ↑	1.91% ↑	0.68% ↑
3000	8.52% ↓	8.13% ↑	1.78% ↑	5.31% ↑

小结:数据库的查询性能与查询请求的时间范围呈负相关,即查询时间范围越大,访问的数据量越多,聚合计算的计算任务也越大,导致查询性能变差,对于后续的实验,将采用一个中间值 1800s 作为系统查询的时间范围配置.

表5、表6展示了在默认条件下不同的设备数和传感器数对于 IoTDB 查询性能的影响,随着设备数和传感器数的增多,数据库查询的数据量呈现线性递增,相应的平均延迟逐渐

表 5　查询的设备数变化对查询的影响,传感器数量为 10

Table 5　Impact of changes in the number of devices queried on the query with a sensor count of 10

Device Number	Throughput (Point/Sec)	Average Latency(ms)	Average CPU Usage(%)	I/O Amount (GB)
1	110.95% ↑	91.29% ↓	7.99% ↓	90.38% ↓
5	140.53	5562.66	53.43	87.06
10	9.64% ↓	121.59% ↑	0.45% ↑	119.41% ↑
15	13.64% ↓	247.41% ↑	1.57% ↑	237.61% ↑
20	30.61% ↓	448.01% ↑	14.56% ↑	357.49% ↑
25	51.52% ↓	732.55% ↑	16.24% ↑	465.45% ↑

变大,而针对吞吐量这一指标,二者表现则略有不同,基准线为传感器数量为 10、设备数量为 5. 随着 Device 数量的增多,系统的吞吐量逐渐降低,但随着 Sensor 数量的增多,系统吞吐量却反而逐渐提升. 例如,当查询的 Sensor 数为 20 时,相对于 Sensor 数为 1 时,系统吞吐量提升了 18.1%,平均延迟增加了 18.7 倍,I/O 数据总量增加了 16.8 倍. 而当查询的 Device 数为 20 时,相对于 Device 数为 1 时,系统吞吐量降低了 67.1%,平均延迟增加了 62.9 倍,I/O 的数据总量增加了 47.6倍. 这是因为查询的基本单位是一个 Page,而一个 Page 中则包含着同一个设备所有传感器的一系列数据,所以当查询的 Sensor 数量增加时,一次查询能够在一个 Page 中查询到更多的数据点,能够提升系统吞吐量. 但当查询的 Device 数量增加时,涉及到的 Page 数量就会增加,I/O 访问次数更多,这会导致查询操作的时长增加,虽然查询到的数据量增加,但查询时长增速更快,这就导致系统吞吐量的下降,这与 Page 的测试研究相关,详情请参考 3.2 节.

表 6 查询的传感器数变化对查询的影响,设备数量为 5

Table 6 Impact of changes in the number of sensors queried on the query with a device count of 5

Device Number	Throughput (Point/Sec)	Average Latency(ms)	Average CPU Usage(%)	I/O Amount (GB)
1	13.91% ↓	89.17% ↓	5.06% ↓	87.95% ↓
5	2.46% ↓	48.52% ↓	2.81% ↑	48.45% ↓
10	141.55	5473.96	53.36	86.75
15	0.91% ↑	51.93% ↑	0.91% ↓	51.74% ↑
20	1.61% ↑	102.91% ↑	0.82% ↓	102.21% ↑
25	2.59% ↑	147.31% ↑	1.12% ↑	140.96% ↑

小结:随着每条查询涉及到的 Device 数和 Sensor 数的增多,系统访问的 I/O 数据量也随之增多,这会导致 IoTDB 系统的延迟呈线性递增趋势. 值得注意的是,当传感器数量增加时,系统的吞吐量是上升的,而设备数增多时,系统的吞吐量是逐渐降低的.

在实际的 IoT 场景下,时序数据常常会出现乱序的情况,为了探究乱序数据占比对 IoTDB 读写性能的影响,本文设计了共 11 组实验,将乱序占比分别设置为 0% 到 100% 进行测试研究. 从图 2 可以观察到,随着乱序数据占比的增加,IoTDB 的写性能基本上稳定在平均每秒钟写入 8875230 个数据点,尽管呈现出一定的波动性,这主要归功于 IoTDB 的 LSM-Tree 架构,其针对大规模数据的写入进行优化,将更多的随机小写,转化为顺序的 batch 操作,并通过不同层级的数据整合 Compaction 操作,有效地提升了数据的写性能. 而对于数据的查询操作,系统吞吐量随着乱序比的增多呈现线性下降,平均延迟呈线性增加. 主要原因是在顺序存储的情况下,IoTDB 可以进行更加高效的二分查找,而随着乱序比例的上升,IoTDB 不得不对每层的数据进行线性搜索,查询操作产生的 I/O 数据总量也在不断增多,从而极大地降低了查询效率. 从图 5 中的数据可以看出,当乱序比为 30% 时,相对于乱序比为 0% 的情况,系统吞吐量降低了 53.1%,平均延迟增加了2.1倍,I/O 数据总量也相应地增加了 2.5 倍.

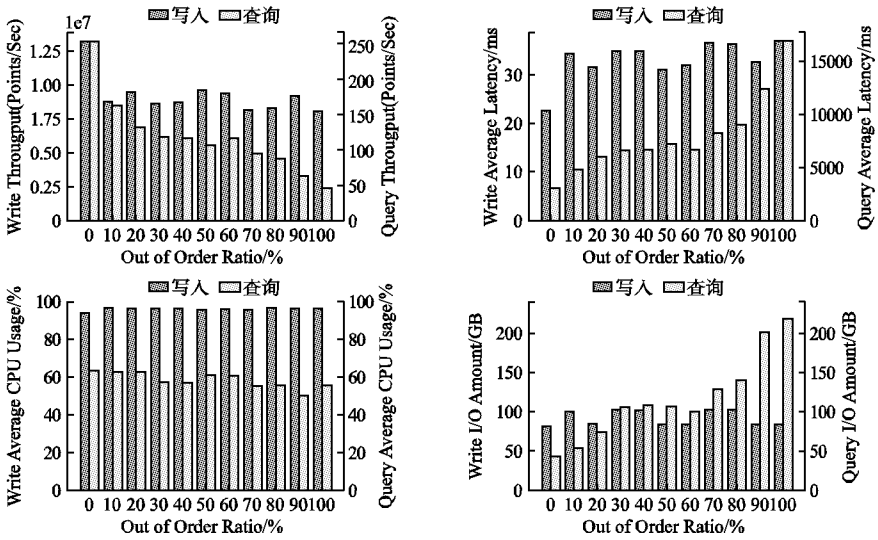

图 2 不同乱序比对 IoTDB 性能的影响

Fig. 2 Impact of different ratios of out-of-order data on IoTDB performance

小结:乱序数据会在一定程度上降低 IoTDB 的读写性能,但得益于 LSM-Tree 结构对写操作的优化,IoTDB 的写性能在不同乱序数据占比的情况下都能保持相对稳定,对于查询而言,乱序数据占比越大,会导致数据搜索产生更多 I/O 操作,查询效率显著下降. 这突显了在乱序数据环境下,IoTDB 的查询操作可能会受到更大的挑战,需要进一步的性能优化和适应策略.

表 7 展示了不同数据类型对于 IoTDB 性能的影响,基准线是数据类型为 DOUBLE 时的性能数据. 从整体上看,DOUBLE 类型数据和 INT64 类型数据性的读写性能相当,这种趋势也可以体现在 I/O 数据总量上. 相比与另外两种数据类型 FLOAT 和 INT32,DOUBLE 和 INT64 类型的数据导致系统的整体性能都有显著下降,I/O 数据总量明显增多. 举例而言,相比于 FLOAT 类型数据,DOUBLE 类型数据造成系统

写吞吐量下降了 14.1%,延迟增加 16.1%,I/O 数据总量增加了 82.4%;就查询性能而言,系统吞吐量降低了 67.5%,平均延迟增加了 3.2 倍,而相应的 I/O 数据总量增 67.5%,平均延迟增加了 3.2 倍,而相应的 I/O 数据总量增了 4.7 倍. 实验结果表明,IoTDB 对不同类型的数据进行读写操作时,由于占用的字节数不同,会产生不同的 I/O 数据量,这也是造成性能不同的主要原因. 此外,不同类型的数据也会影响数据的编码、压缩效率,本文在 3.2 节对此展开了深入研究.

表 7　不同类型数据在数据库中读写性能表

Table 7　Table of read and write performance of different types of data in the database

Data Type	W Throughput (Point/Sec)	W AVG Latency(ms)	W AVG CPU Usage(%)	W I/O Amount(GB)	Q Throughput (Point/Sec)	Q AVG Latency(ms)	Q AVG CPU Usage(%)	Q I/O Amount(GB)
INT32	23.02% ↑	19.07% ↓	2.19% ↓	58.68% ↓	371.48%	80.15% ↓	11.03% ↑	86.05% ↓
INT64	1.84% ↓	1.98% ↑	0.36% ↑	0.53% ↓	11.16% ↓	2.87% ↑	3.26% ↓	9.17% ↑
FLOAT	15.79% ↑	13.77% ↓	5.28% ↓	46.16% ↓	207.44%	68.63% ↓	15.03% ↑	78.88% ↓
DOUBLE	13260801.71	22.66	94.22	180.45	252.39	3099.69	63.35	42.52

小结:IoTDB 对不同类型的数据进行读写操作时,由于不同类型数据所占的字节数不同,系统产生了不同的 I/O 数据总量,这是导致数据库性能差异的主要原因.

3.2　数据库内部组件对于 IoTDB 性能的影响

为探索数据库内部组件对 IoTDB 性能的影响,本文选取了 Page Size、整合策略、编码方式、压缩算法这 4 个重要模块进行测试.

Page 作为 IoTDB 数据读写的基本单位,本文研究了不同大小的 Page Size 对数据库读写性能的影响,将该参数设为 1KB、4KB、16KB、64KB、256KB 进行实验,该部分实验数据的基准线为 Page 选取 64 KB 时的数据. 从表 8 和表 9 中可以

表 8　Page 大小对 IoTDB 写入性能的影响

Table 8　Impact of page size on IoTDB write performance

Page Size	Throughput (Point/Sec)	Average Latency(ms)	Average CPU Usage(%)	I/O Amount (GB)
1KB	3.73% ↓	3.29% ↑	7.94% ↓	0.44% ↑
4KB	11.92% ↓	13.19% ↑	11.96% ↓	2.21% ↑
16KB	0.37% ↑	0.67% ↓	0.36% ↓	0.14% ↓
64KB	12682495.63	23.73	93.88	180.06
256KB	10.71% ↓	12.27% ↑	12.54% ↓	4.56% ↓

发现,相同的操作,如果 Page Size 设置太小,读写的数据会分散到更多的 Page 中,从而产生更多的 I/O 操作,影响性能;相反,如果 Page Size 设置过大,会引起较大的读写放大问题,也

表 9　Page 大小对 IoTDB 查询性能的影响

Table 9　Impact of page size on IoTDB query performance

Page Size	Throughput (Point/Sec)	Average Latency(ms)	Average CPU Usage(%)	I/O Amount (GB)
1KB	5.96% ↓	6.64% ↑	0.45% ↓	8.39% ↑
4KB	9.52% ↓	10.59% ↑	0.92% ↓	13.51% ↑
16KB	8.46% ↓	9.31% ↑	3.47% ↓	11.91% ↑
64KB	178.76	4363.98	54.21	70.84
256KB	22.61% ↓	29.49% ↑	8.73% ↓	36.61% ↑

会造成系统性能严重下降. 例如,相对于 64KB Page Size,设置为 1KB 的情况下,就查询性能而言,IoTDB 系统吞吐量降低了 5.9%,平均延迟增加了 6.6%,读写 I/O 总量增加了 8.4%;与 256KB 的 Page Size 相比,IoTDB 的查询系统吞吐量减低了 22.6%,平均延迟增加了 29.5%,读写 I/O 总量增加了 36.6%.

小结:过小的 Page Size 会将数据读写分散到更多的 Page,而过大的 Page Size 会造成严重的读写放大问题,因此根据不同工作负载的情况,合理设置 Page Size 对于系统的性能至关重要. 在本次实验环境下,将 Page Size 设置为 64KB 时,能够更好地匹配上层时序数据工作负载的特征,保证数据库有较好的读写性能,但真实的环境下,时序数据工作负载可能是动态变化的,如何动态调整 Page Size 以更好地适应上层工作负载可能是一个非常重要的研究方向.

IoTDB 在进行数据整合时,会涉及到乱序空间内数据整合、顺序空间内数据整合和跨空间的数据整合. 不同的数据整合策略会对 IoTDB 的性能产生重要的影响,IoTDB 提供了 3 种不同的策略:1) BALANCE 平等对待各种整合顺序;2) INNER_CROSS 优先进行顺序 TsFile 和顺序 TsFile 或乱序 TsFile 和乱序 TsFile 的整合;3) CROSS_INNER 优先将乱序 TsFile 整合到顺序 TsFile 中. 为了验证不同整合策略对数据库性能的影响,本文将对不同乱序占比(10%、50%、90%)的数据进行写入和查询测试,写入的 loop 设置为 5000.

从图 3 可以观察到,可以发现当乱序数据占比为 10% 时,在写入的过程中,采用 CROSS_INNER 整合策略的实验的性能相对于另外两种合并策略更优,这是因为当乱序比例比较小时,乱序数据占比小,CROSS_INNER 优先的跨空间合并操作少,更多的系统资源向数据写入倾斜,例如 CROSS_INNER 整合策略产生的 I/O 总量比 BALANCE 和 INNER_CROSS 整合策略分别减少了 2.6% 和 1.8%,但对数据查询而言,INNER_CROSS 策略下的查询性能最好,这是因为在该策略下,优先顺序空间和乱序空间内的整合操作,减少了文件数量,查询时减少了需要进行遍历的文件数量,这也反映在读写 I/O 数据总量上,相比与 BALANCE 和 CROSS_INNER 整合策略减少了 9.3% 和 23.7%;但随着乱序数据占比的增加,例如乱序数据占比为 50%,BALANCE 策略在写入方面性能更好,这是由于 BALANCE 同时关注于数据文件总量减少和乱序文件减少,能够平衡写入操作和合并操作的系统资源分配,提升数据库写性能,就读写 I/O 总量而言,相比与INNER_CROSS 和 CROSS_INNER 整合策略分别减少 18.3% 和 0.4%;值得注意的是,当乱序数据占比为 90%,CROSS_INNER 策略下查询性能最佳,因为随着乱序数据占据绝大多数,减少乱序文件的占比能够提升数据查询的性能,从 IO 可

以看出 CROSS_INNER 策略相对 BALANCE 策略在读写总量上减少了 13.5%,相对 INNER_CROSS 策略减少了 7.3%.

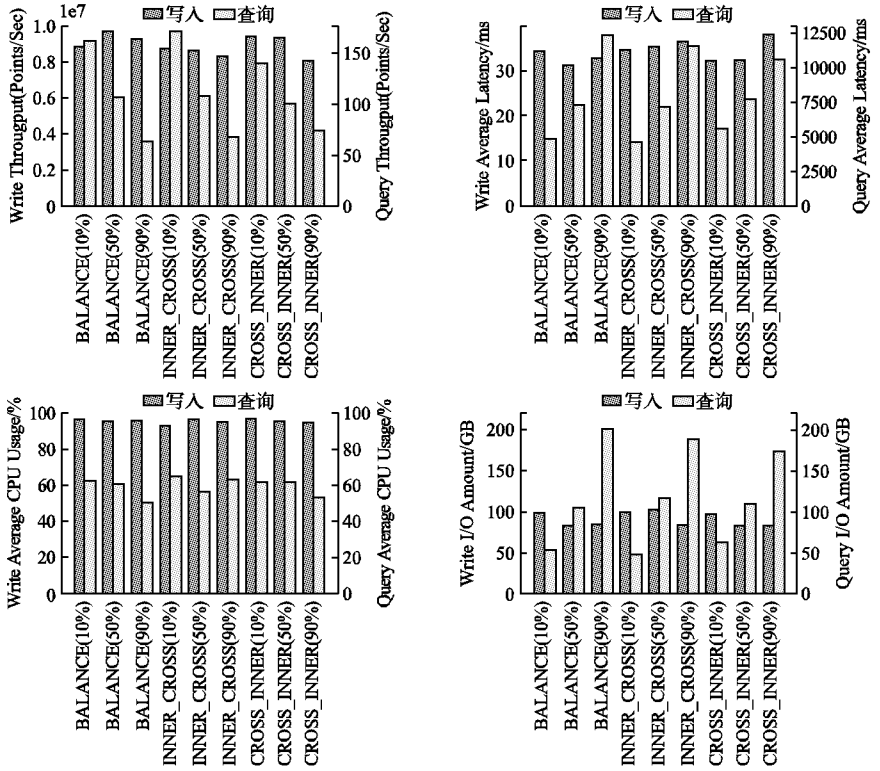

图 3 不同合并策略对 IoTDB 性能影响

Fig. 3 Impact of different compaction strategies on IoTDB performance

小结:当乱序数据占比较小时,使用 CROSS_INNER 策略,数据库的写性能最好,读性能较差. 但随着乱序数据增加时,BALANCE 策略能提升数据库的写性能,CROSS_INNER 策略能提升数据库的读性能. 所以根据实际的工作负载,动态选择对应的整合策略,能显著提升 IoTDB 的读写性能.

IoTDB 对于时序数据的压缩可以分成两个关键步骤:首先时序数据经过编码处理转换成二进制流,再对这些二进制流进行压缩操作. 在本次实验中,将数据编码和压缩算法拆分程两个独立的模块分别进行测试,以探究其各自对 IoTDB 的性能影响. IoTDB 支持不同的编码方式和压缩算法,主要包括以下几种:

- RLE(Run-Length Encoding):使用行程长度编码,将相邻的相同数值用一个数值和一个计数表示,以减小数据存储空间;
- TS_2DIFF(Two-Step Second-Order Difference):对时序数据进行二阶差分编码,以减小时间戳数据的存储量,提高存储效率;
- Gorilla:一种专门设计用于时序数据的编码方式,具有高效的压缩性能,尤其适用于稀疏的时序数据;
- SNAPPY:一种快速压缩算法,旨在提供较高的压缩速度和适度的压缩比;
- LZ4:一种具有极高压缩和解压速度的算法,适用于需要快速处理的场景;
- GZIP:一种通用的压缩算法,提供较高的压缩比,但相对而言速度较慢.

表 10 编码方式对数据库读写性能的提升明细,基准线为不采用编码方式(PLAIN)

Table 10 Breakdown of database read and write performance improvement by encoding method, baseline is without encoding method(PLAIN)

性能指标	RLE	TS_2DIFF	DORILLA
W Throughput	8.46% ↑	16.53% ↑	8.59% ↑
W AVG Latency	7.85% ↓	14.12% ↓	7.72% ↓
W AVG CPU Usage	0.64% ↓	1.41% ↓	0.01% ↓
W I/O Amount	40.18% ↓	43.43% ↓	29.59% ↓
Q Throughput	252.99% ↑	318.14% ↑	141.73% ↑
Q AVG Latency	72.96% ↓	77.47% ↓	59.71% ↓
Q AVG CPU Usage	1.28% ↓	1.35% ↓	5.84% ↑
Q I/O Amount	88.61% ↓	91.36% ↓	77.41% ↓

图 4 清晰地展示了不同的编码方式对 IoTDB 读写性能的有效提升同时表 10 清晰的列出了不同编码方式对 IoTDB 读写性能影响的具体数据. 就写性能而言,与 PLAIN 编码方式相比,RLE 在吞吐量上提升了 8.5%,平均延迟降低了 7.9%,数据读写总量降低 40.2%,TS_2DIFF 在吞吐量上提升了 16.5%,平均延迟降低了 14.1%,数据读写总量降低 43.4%,Gorilla 在吞吐量上提升了 8.6%,平均延迟降低了 7.7%,数据读写总量降低 29.6%. 在查询性能方面,不同的编码方式也能够显著提升其性能. 相对于 PLAIN 编码,RLE 在吞吐量上提升了 2.5 倍,平均延迟降低了 73.0%,数据读

写总量降低 88.6%,TS_2DIFF 在吞吐量上提升了 3.2 倍,平均延迟降低了 77.5%,Gorilla 在吞吐量上提升了 1.4 倍,平均延迟降低了 59.7%,数据读写总量降低 91.4%. 整体来看,TS_2DIFF 编码方式表现最佳,主要是因为对时序数据编码可以有效提升单个 Page 的存储效率,从而在相同的数据操作中有效地减少其 I/O 访问数据总量,显著提升访问性能. 同时,时序数据编码还能有效降低存储空间,节省存储成本,这对于快速增长的时序数据规模是至关重要的. 在实验中发现,相对于 PLAIN 编码方式,RLE、TS_2DIFF 和 Gorilla 分别减少存储空间消耗 79.8%、84.7% 和 58.9%.

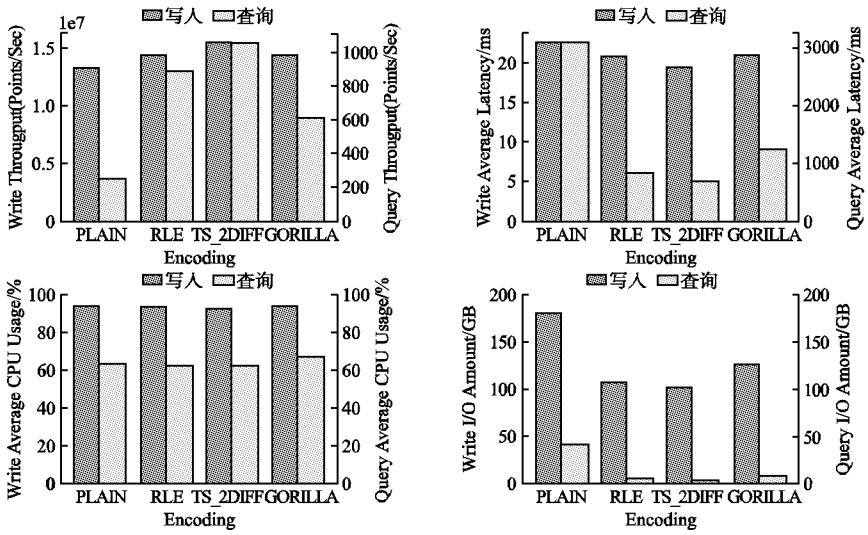

图 4　不同的编码方式对 IoTDB 性能的影响

Fig. 4　Impact of different encoding methods on IoTDB performance

除了时序数据编码之外,不同的压缩算法也能在数据访问性能和存储效率方面带来益处,图 5 显示了不同压缩算法对数据库读写性能的影响同时表 11 清晰的列出了不同压缩算法对 IoTDB 读写性能影响的具体数据. 相对于 UNCOMPRESSED,使用 SNAPPY 压缩算法对数据进行写入、查询时分别在吞吐量上提升 17.0%、4.1 倍,平均延迟降低了 14.2%、81.6%,数据读写总量降低了 54.7%、97.7%,对数据进行查询时,CPU 使用率提高了 37.9%;使用 LZ4 压缩算法对数据进行写入、查询时分别在吞吐量上提升 16.0%、4.1 倍,平均延迟降低了 13.6%、81.6%,数据读写总量降低了 99.2%、

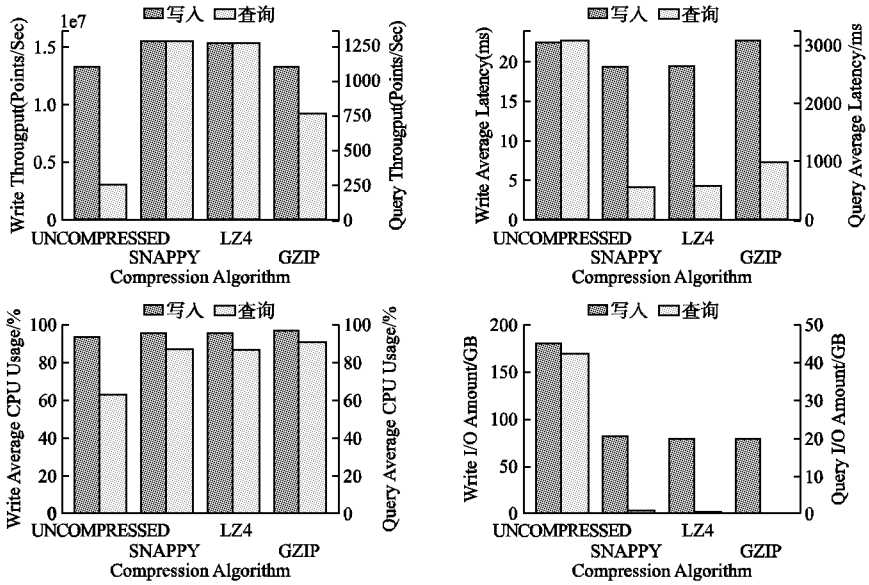

图 5　压缩算法对 IoTDB 性能的影响

Fig. 5　Impact of compression algorithms on IoTDB performance

55.5%,对数据进行查询时,CPU 使用率提高 36.7%;使用 GZIP 压缩算法对数据进行写入、查询时分别在吞吐量上提升 0.2%、2.0 倍,平均延迟在写入上提升了 0.9%、在查询时降低了 68.1%,数据读写总量降低了 55.6%、99.3%,对数据进

行查询时,CPU 使用率提高 43.5 个百分点. 在存储空间优化方面,相对于 UNCOMPRESSED 方式,SNAPPY、*LZ4*、GZIP 分别减少了存储空间消耗 97.3%、97.4% 和 97.8%.

小结:在时序数据库管理系统中,选择合适的编码和压缩算法可以显著提高数据库的访问性能,并在存储方面实现更高效的利用. 综合本次实验结果发现,TS_2DIFF 更适合时序数据编码,*LZ4* 和 SNAPPY 压缩算法相较于 GZIP 更适合作为时序数据压缩.

表 11 压缩算法对数据库读写性能的提升明细,基准线为不使用压缩算法(UNCOMPRESSED)

Table 11 Breakdown of database read and write performance improvement by compression algorithm, baseline is without compression algorithm (UNCOMPRESSED)

性能指标	SNAPPY	LZ4	GZIP
W Throughput	16.99% ↑	16.01% ↑	0.17% ↑
W AVG Latency	14.21% ↓	13.54% ↓	0.92% ↓
W AVG CPU Usage	1.89% ↑	1.73% ↑	2.75% ↑
W I/O Amount	54.65% ↓	55.52% ↓	55.59% ↓
Q Throughput	411.61% ↑	405.56% ↑	201.85% ↑
Q AVG Latency	81.85% ↓	81.56% ↓	68.12% ↓
Q AVG CPU Usage	37.92% ↑	36.71% ↑	43.51% ↑
Q I/O Amount	97.71% ↓	99.22% ↓	99.34% ↓

3.3 底层存储设备对于数据库性能的影响

近年来,新兴存储设备发展迅猛,包括读写速度更快的闪存 SSD 以及通过磁道叠加实现存储密度更高的 SMR,正在重新塑造底层存储系统的格局. 为探究不同底层存储设备对 IoTDB 读写性能的影响,本文选取了 3 种代表性的存储设备,即 SSD、HDD、SMR,进行实验研究,3 种设备的基本信息可以参考表 1.

观察表 12 发现底层存储设备对于 IoTDB 的性能有着显著的影响,就写性能而言,SSD 相对于 HDD 和 SMR 在系统吞吐量上分别提升了 4.1 倍和 11.1 倍,平均时延分别降低了 80.5% 和 91.6%;在查询性能方面,SSD 相对于 HDD 和 SMR 在系统吞吐量上分别提升了 7.7 倍和 16.7 倍,平均时延分别降低了 88.5% 和 92.4%. 主要原因是因为 SSD 具有更高的访问带宽和更低的访问延迟. 类似地,也可以发现 HDD 在读写吞吐量和访问延迟方面表现比 SMR 更为优越.

表 12 IoTDB 使用不同底层存储设备时的读写性能

Table 12 Read and write performance of IoTDB when using different underlying storage devices

性能指标	SSD	HDD	SMR
W Throughput (Point/Sec)	13260802.71	2577494.95	1098769.21
W AVG Latency (ms)	22.66	116.02	270.82
W AVG CPU Usage (%)	94.22	21.05	9.91
W I/O Amount (GB)	180.44	201.98	201.27
Q Throughput (Point/Sec)	252.39	29.07	14.21
Q AVG Latency (ms)	3099.69	27041.78	55581.86
Q AVG CPU Usage (%)	63.34	9.57	5.41
Q I/O Amount (GB)	42.52	81.78	83.84

小结:底层存储设备对于 IoTDB 的读写性能有着显著的影响,尽管 SSD 可以有效提升数据库的访问性能,但 SMR 这种价格低廉、存储密度高的新型设备可以有效降低时序数据存储成本,尤其是考虑 IoT 时代,时序数据呈现爆炸式快速增长的趋势. 因此,如何将两者融合构建时序数据混合存储系统,充分考虑性能和成本之间的权衡,为应对大规模时序数据的挑战提供了一种有效的解决方案.

3.4 多性能指标及参数相关性分析

对 IoTDB 多性能指标进行相关性分析是数据库性能表现的一个重要方式,通过计算相关系数来测量目标之间的关联程度,然后利用可视化工具-热力图,将相关性结果进行可视化展示,以便更直观地理解不同目标之间的联系.

如表 13 所示,在计算不同指标的相关性时,需要对数据库参数进行分类处理. 为此,本文对研究涉及的参数进行了重新命名,以更加清晰地呈现不同调节旋钮与性能指标之间的关联性. 例如,CN-Client Number 表示执行读写操作的客户端数量,该参数为数值型变量,可直接计算相关性系数. 而 CP-Compressor 变量用于指代数据压缩时所采用的压缩算法,当选用 *LZ4* 压缩算法时,该变量名将变更为 CP-*LZ4*. 由于该类参数为非数值型变量,因此在计算相关性系数之前,需要对其进行适当的编码转换.

表 13 系统参数介绍

Table 13 Introduction of system parameters

参数英文及其缩写	参数含义
CN-Client Number	客户端数量
CP-Compressor	压缩算法
DT-Data Type	数据类型
DV-Device	底层存储设备
EC-Encoding	编码方式
PS-Page Size	页面大小
PO-Poisson	乱序比
PR-Priority	合并优先级
QR-Query	查询时间范围
QS-Query Sensor	查询涉及的传感器数量
QD-Query Device	查询涉及的设备数量

图 6 展示了通过对 IoTDB 的多个性能指标以及参数进行相关性分析之后进行热图展示的结果,通过相关性系数公式计算出的相关性系数数 γ 的取值范围在 $[-1,1]$ 之间,数 γ 的值越大表示相关程度越高,通常认为相关性系数绝对值大于 0.8 表示两者之间存在强相关性,而相关性系数接近 0,则表示两个变量之间彼此独立,没有相关性. 相关性系数计算公式如下,两个变量 X 和 Y 的均值,分别记为 $\bar{X}$ 和 $\bar{Y}$,X_i 和 Y_i 分别代表两个变量 X 和 Y 第 i 个值:

$$r=\frac{\sum(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum(X_i-\bar{X})^2(Y_i-\bar{Y})^2}}$$

如图 7 所示,对于数值型数据,如吞吐量、延迟等,可按照计算公式直接计算其相关性系数. 对于非数值型参数,将分类变量转换为虚拟变量,具体步骤如下:首先,确定属于分类变量的参数,如压缩算法、底层存储设备、编码方式等参数属于分类变量;其次,使用独热编码对分类变量,为每一个类别创建一个新的二进制列,这些新列中的值是 0 或 1,代表原始分类变量中的类别,例如,对于底层存储设备这个分类变量,它

的值可以是 HDD、SMR 或 SSD,将这个变量转换为虚拟变量后,将得到 3 个新的二进制列:DV_HDD、DV_SMR 和 DV_SSD,如果观察到数据中某行的底层存储设备的值是 HDD,那么 DV_HDD 列的值将是 1,而 DV_SMR 和 DV_SSD 的值将是 0;最后,按照公式计算性能指标与虚拟变量之间的相关性系数,可以得到不同分类变量对性能的影响趋势.

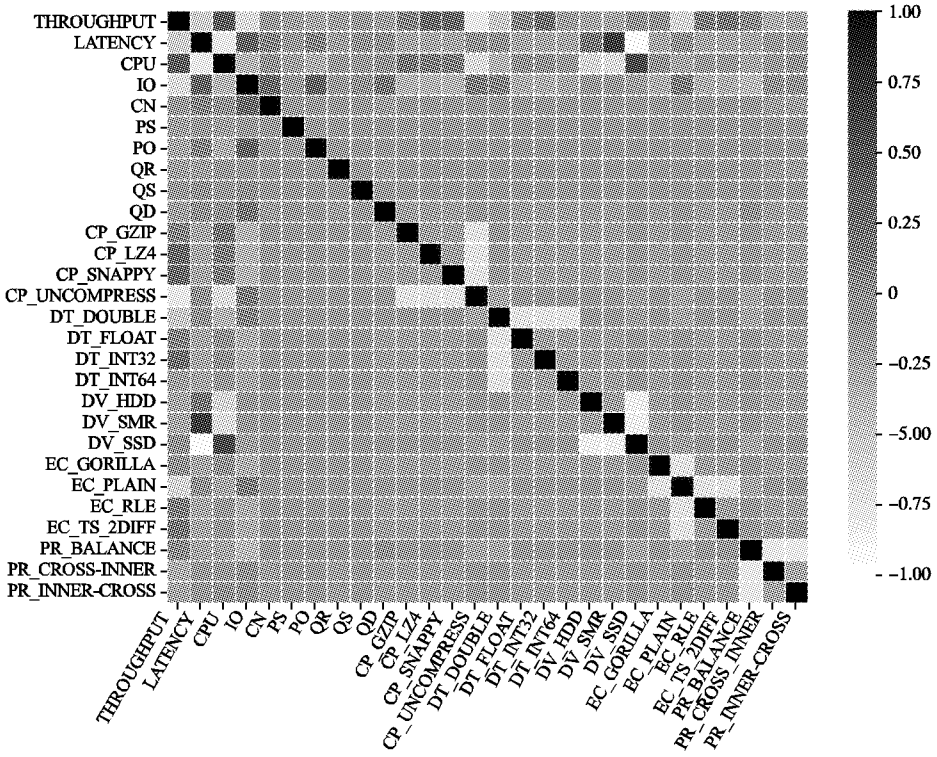

图 6　系统指标与参数相关性分析

Fig. 6　Correlation analysis of system indicators and parameters

从图 6 中,可以发现从左上角到右下角有一条黑色的对角线,显示为强相关性,这是因为对角线指向的双方都是同一参数或指标. 对于系统的吞吐量而言,该指标与 CPU 使用率指标呈正相关,与平均时延和读写量大小为负相关,而CPU使用率与平均延迟呈强负相关. 从观察中可发现,对数据进行编码和压缩对于提升数据库性能都是正相关的,能大幅提高数据库的读写性能. 同时采用不同的底层设备对于数据库性能的影响也十分重要,如使用 SSD 作为底层存储设备,可以大幅度减少系统的读写时延. 在不同类型的数据写入数据库时,对数据库的性能也有影响,写入的数据所占字节数越小,对应的数据库的读写性能也就越好. 同时还可以从图中观察到,QD(查询涉及的设备数量)与读写量大小呈强正相关,增加查询涉及的设备数量能够大大提升读写量大小,而增加查询涉及的传感器数量却对读写量大小的提升并不明显.

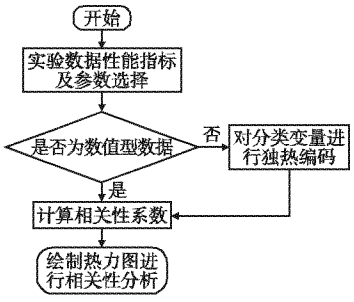

图 7　相关性分析流程图

Fig. 7　Flow chart of correlation analysis

4　总　结

本文从系统研究的角度出发,深入探究 IoTDB 在时序数据工作负载下的性能表现,覆盖了底层存储设备、数据压缩算法、数据整合策略、上层工作负载等多个关键模块. 通过对以上实验结果的深入分析,揭示了在不同的工作负载下,不同的参数选择对数据库性能的差异影响,证实了对写入的数据进行编码和压缩能显著提升数据库性能,同时采用速度更快的存储设备也能有效提升数据库的读写性能. 此外,本文还对实验涉及到的性能指标和实验参数进行了相关性分析,为研究者和从业者提供了更深层次理解时序数据库性能特征的指导,有望促进该领域的进一步创新和优化.

References:

[1] Bhandari S, Bergmann N, Jurdak R, et al. Time series analysis for spatial node selection in environment monitoring sensor networks [J]. Sensors, 2017, 18(1): 1-16.

[2] Roberts L, Michalak P, Heaps S, et al. Automating the placement of time series models for iot healthcare applications [C]//14th International Conference on e-Science (e-Science), 2018: 290-291.

[3] Roesener C, Fahrenkrog F, Uhlig A, et al. A scenario-based assessment approach for automated driving by using time series classifica-

tion of human-driving behaviour [C]//IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), 2016: 1360-1365.

[4] Andersen M P, Culler D E. Btrdb: optimizing storage system design for timeseries processing [C]//Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16), 2016: 39-52.

[5] WANG J, DENG S M, ZHOU K J, et al. Implementation of database testing tool based on TPC-DS [J]. Laboratory Science, 2021, 24(1): 25-28.

[6] Nasar M, Kausar M A. Suitability of influxdb database for iot applications [J]. International Journal of Innovative Technology and Exploring Engineering, 2019, 8(10): 1850-1857.

[7] Wang C, Huang X, Qiao J, et al. Apache IoTDB: Time-series database for internet of things [J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2901-2904.

[8] Petre I, Boncea R, Radulescu C Z, et al. A time-series database analysis based on a multi-attribute maturity model [J]. Studies in Informatics and Control, 2019, 28(2): 177-188.

[9] An S Y, Cha Y S, Jeon E J, et al. A pre-study on the open source prometheus monitoring system [J]. Smart Media Journal, 2021, 10(2): 110-118.

[10] DU L J. Performance comparison between traditional relational database and NoSQL database [J]. Intelligent Computer and Applications, 2017, 7(3): 132-134.

[11] Adams C, Alonso L, et al. Monarch: Google's planetscale in-memory time series database [J]. Proceedings of the VLDB Endowment, 2020, 13(12): 3181-3194.

[12] Agrawal N, Vulimiri A. Low-latency analytics on colossal data streams with summarystore [C]//Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), 2017: 647-664.

[13] Liu J, Wang K, Chen F. TSCache: an efficient flash-based caching scheme for time-series data workloads [J]. Proceedings of the VLDB Endowment, 2021, 14(13): 3253-3266.

[14] Pelkonen T, Franklin S, Teller J, et al. Gorilla: a fast, scalable, in-memory time series database [J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1816-1827.

[15] Liu R, Yuan J. Benchmarking time series databases with IoTDB-benchmark for IoT scenarios [J]. arXiv preprint arXiv: 1901.08304, 2019.

[16] Shi X, Feng Z, Li K, et al. Byteseries: an in-memory time series database for large-scale monitoring systems [C]//Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC), 2020: 60-73.

[17] Wang C, Qiao J, Huang X, et al. Apache iotdb: a time series database for iot applications [J]. Proceedings of the ACM on Management of Data, 2023, 1(2): 1-27.

[18] Hao Y, Qin X, Chen Y, et al. TS-Benchmark: a benchmark for time series databases [C]//IEEE 37th International Conference on Data Engineering (ICDE), 2021: 588-599.

[19] Jia Y, Chen F. Kill two birds with one stone: auto-tuning rocksdb for high bandwidth and low latency [C]//IEEE 40th International Conference on Distributed Computing Systems (ICDCS), 2020: 652-664.

[20] Liu J, Wang K, Chen F. Understanding energy efficiency of databases on single board computers for edge computing [C]//29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2021: 1-8.

[21] Raasveldt M, Holanda P, Gubner T, et al. Fair benchmarking considered difficult: common pitfalls in database performance testing [C]//Proceedings of the Workshop on Testing Database Systems, 2018: 1-6.

[22] Tongkaw S, Tongkaw A. A comparison of database performance of MariaDB and MySQL with OLTP workload [C]//IEEE Conference on Open Systems (ICOS), 2016: 117-119.

[23] Ray S, Simion B, Brown A D. Jackpine: a benchmark to evaluate spatial database performance [C]//IEEE 27th International Conference on Data Engineering (ICDE), 2011: 1139-1150.

附中文参考文献：

[5] 王 洁，邓双敏，周宽久，等. 基于 TPC-DS 的数据库测试工具实现 [J]. 实验室科学，2021，24(1)：25-28.

[10] 杜丽娟. 关系型数据库与 NoSQL 数据库的性能对比 [J]. 智能计算机与应用，2017，7(3)：132-134.