arXiv:2301.12561v1 [cs.DB] 29 Jan 2023

高频数据专用数据库基准测试

预印本

Fazl Barez爱丁堡大学爱丁堡机器人中心
f.barez@ed.ac.uk

Paul Bilokon伦敦帝国理工学院计算机系
paul.bilokon@imperial.ac.uk

Ruijie Xiong伦敦帝国理工学院计算机系
ruijie.xiong21@imperial.ac.uk

2023年1月31日

摘要

本文提出了一套基准测试套件，用于评估和比较面向高频数据的时间序列数据库，并重点关注金融应用。该套件包含四种专用数据库：ClickHouse、InfluxDB、kdb+ 和 TimescaleDB。测试结果表明，在受测数据库中，kdb+ 的性能最高，同时也揭示了各数据库的优势与不足。该基准测试套件旨在客观衡量这些数据库的性能，并比较它们处理不同类型数据的能力。这为评估不同时间序列数据库是否适用于不同用例提供了宝贵见解，也提供了可用于指导系统设计决策的基准测试。

1 引言

1.1 动机

在大数据时代，金融和技术等各行各业每天都会产生海量时间序列数据。时间序列数据具有数据量大、更新频率高于传统关系型数据等特点，这意味着新数据点不断产生，而旧数据很少更新。因此，传统数据库系统需要经过优化，才能高效存储和操作这些时间序列数据。为了满足现实场景中对读写吞吐量和延迟的严格要求，需要专用时间序列数据库。

常见的高频数据专用数据库包括 InfluxDB 和 Kdb+。这两种数据库都提供一系列功能，用于时间序列数据的存储、管理和分析。InfluxDB 用于存储和处理时间序列数据，并提供强大的查询语言以检索数据。Kdb+ 是一种列式数据库，通过其内存数据库、查询语言和分析平台支持高频数据分析。

专用数据库正逐渐成为满足时间序列数据特定特征需求的关键，并且在需要进行高频数据管理和分析的应用中越来越受欢迎。

为公司选择合适的数据库，需要借助基准测试来有效评估和比较不同的时间序列数据库。传统的关系型数据库基准测试（例如 TPC 基准测试） [1],并不适用于大规模时间序列数据应用。因此，STAC-M3 基准测试套件 [2] 专为金融逐笔数据分析而开发，但目前仅应用于少数数据库，例如 kdb+和 eXtremeDB。

Cooper 等人 [3] 提出了面向大数据应用的 YCSB 开源基准测试套件，但它不适用于时间序列数据库的典型工作负载。为弥补这一空缺，我们提出了一套适用于金融领域时间序列数据的专用开源基准测试套件。该套件使用加密货币交易所数据模拟工作负载，并用于评估 ClickHouse、InfluxDB、kdb+和 TimescaleDB。

2 文献综述

2.1 时间序列数据库

时间序列数据具有不可变性、持续动态更新和规模通常较大的特点。这些通常源于金融应用的数据不同于传统数据（例如关系表），可用于分析和数据挖掘，且规模通常较大 [4]。不可变性也是时间序列数据的一个属性：新数据点总是追加到末尾，而已记录的数据很少被修改。

传统数据库具有冗余功能，且并未针对时间序列数据操作进行优化，例如任意更新或删除存储的数据 [5]。此外，时间序列数据分析通常涉及特定时间范围内的数据，而非整个数据集；与较旧的数据相比，近期数据通常被访问得更频繁 [6]。

近期研究指出，由于时间序列数据通常规模庞大，传统数据库在扩展以处理这类数据时可能面临挑战。为提高存储效率，可以采用压缩和删除策略，并针对具有不同特征的数据调整压缩算法，以优化数据 [7]。此外，还可以运用预处理和汇总技术，加快查询处理速度 [5]。

LittleTable [8] 是 Cisco Meraki 开发的分布式关系型数据库，用于存储设备监控和分析所需的时间序列数据。它经过优化，可根据时间戳和属性对数据进行分区，将同一时间范围内收集的同一层级数据存储在磁盘的相邻位置，并实施可配置的生存时间（TTL）策略，自动删除过期数据。这减少了将大量数据存储在主存中的需求。此外，它采用单写入者和仅追加策略，无需锁定或更新先前的数据。LittleTable 还具备容错能力：数据由地理位置最接近用户的服务器提供；如果主服务器发生故障，备用服务器可以通过预写日志重建数据库并接替其工作。

Gorilla [9], Facebook 的高可用内存时间序列数据库采用基于差分的差分的压缩方案，平均可将数据存储量减少 12 倍。该方法通过计算当前数据点与前两个数据点之间的差值来转换采集到的数据点，即使数据点之间的差异很小，结果也会是一个较小的数值。与其他关系型数据库不同，Gorilla 不优先考虑一致性，而是专注于写入可用性和读取性能；由于时间序列数据通常用于聚合分析，不一致导致的任何数据丢失对整体结果的影响都很小。

Monarch [10], 由 Google 开发，是一种支持监控和分析各类服务的多租户内存时序数据库。Monarch 的模式化关系模型包含多种数据类型和丰富的查询语言，以满足用户的不同需求。为实现高效访问，数据按照目标列的字典序存储在分区中；目标列由与每个时间序列相关联的目标列和指标列组成。为减少所需的内存存储量，系统采用了共享时间戳序列以及复杂类型的增量编码等策略。此外，Monarch 还提供集合聚合功能，可在数据存入内存之前，按照用户定义的配置对数据进行聚合。Monarch 专为实现高写入可用性而设计，因此在网络拥塞时会丢弃延迟写入，以优先处理更新的请求。通过允许查询返回部分数据，并在某些服务器无响应时通过软查询截止时间向用户发出提醒，Monarch 确保了较高的读取可用性。

2.1.1 kdb+

Kdb+ [11] 是 KX Systems 开发的一款商用内存时间序列数据库。为支持快速数据摄取和即时查询，kdb+ 按照实时数据库（RDB）、日内数据库（IDB）和历史数据库（HDB）的层级迁移数据 [12]。数据首先存储在内存中的 RDB 中，用户配置的时间间隔内累积的数据会被转移到临时的磁盘上 IDB 中。数据

IDB 中的数据会进一步整理和排序，之后存储在永久的磁盘上 HDB 中。Kdb+ 以列式表示存储数据，即同一列的数据存放在一起，这与传统关系型数据库的面向行架构相反。由于查询通常针对特定属性，这种方式大幅减少了为响应查询而需要访问的数据量。Kdb+ 依赖内置的编程与查询语言，其体积相对较小，约为 800 kb，可存储在中央处理器的 L1/2 缓存中，以实现高速访问 [13]。此外，kdb+ 在数据库中执行计算和筛选，减少了通过网络传输数据以进行计算所带来的开销。

Kdb+ 展现出卓越的性能，并在行业中拥有广泛的应用。经测试，Kdb+ 的速度比 Cassandra 快 29 倍，比 MongoDB 快 38 倍 [14]。在 STAC-M3 基准测试中，Kdb+ 在 Antuco 套件的 17 项基准测试中有 10 项保持纪录，并在 Kanaga 套件的 10 项基准测试中有 9 项保持纪录；STAC-M3 基准测试是金融服务行业的时间序列数据分析基准测试标准[15]。Cobalt 使用 kdb+构建了其交易后基础设施系统，支持每秒 100 万笔交易的吞吐量，并将成本降低最多 80% [16]。一家全球投资银行利用 Google Cloud 和 kdb+ 构建可扩展的数据库，用于实时和历史市场数据分析，每天处理 1 TB 数据 [17]。一家将 AWS S3 与 kdb+ 结合使用的金融科技初创公司，以相当于使用固态硬盘 20% 的成本实现了 10 倍性能 [18]。

内存数据库

近年来，主存存储设备成本的下降，使得内存数据库能够应用于企业应用。内存数据库是指数据主要驻留在主存而非磁盘中的数据库 [19]。由于访问主存的速度比访问磁盘快几个数量级，因此可以实现数据的实时可用性和分析等功能。内存数据库还需要非易失性存储来进行日志记录与恢复，以确保容错能力 [20]。此外，内存数据库还必须采用高效的压缩技术，在减少存储空间的同时不降低性能。内存数据库还可以利用多核中央处理器，通过根据处理器对数据集进行分区来实现并行化。

面向列的布局

kdb+ 采用面向列的数据布局，而非传统关系型数据库 [21]所使用的传统面向行的数据布局。图1比较了这两种存储模型在内存中的存储方式和访问方式。在 kdb+ 中，同一列的数据存储在一起，各列则可以分散存储；而在 SQL 中，同一行的数据存储在一起。如图1所示，面向行布局中的行访问和面向列布局中的列访问都会顺序访问内存，因此最能受益于内存局部性 [22]。面向行布局适用于侧重事务处理并对行进行操作的联机事务处理（OLTP）工作负载。联机分析处理（OLAP）工作负载大多涉及读取操作，并针对表属性执行查询，因此更适合采用面向列的布局。这些 OLAP 访问模式在时间序列数据库中很常见。

	col1	col2	col3
row1	A	a	1
row2	B	b	2
row3	C	c	3
row4	D	d	4

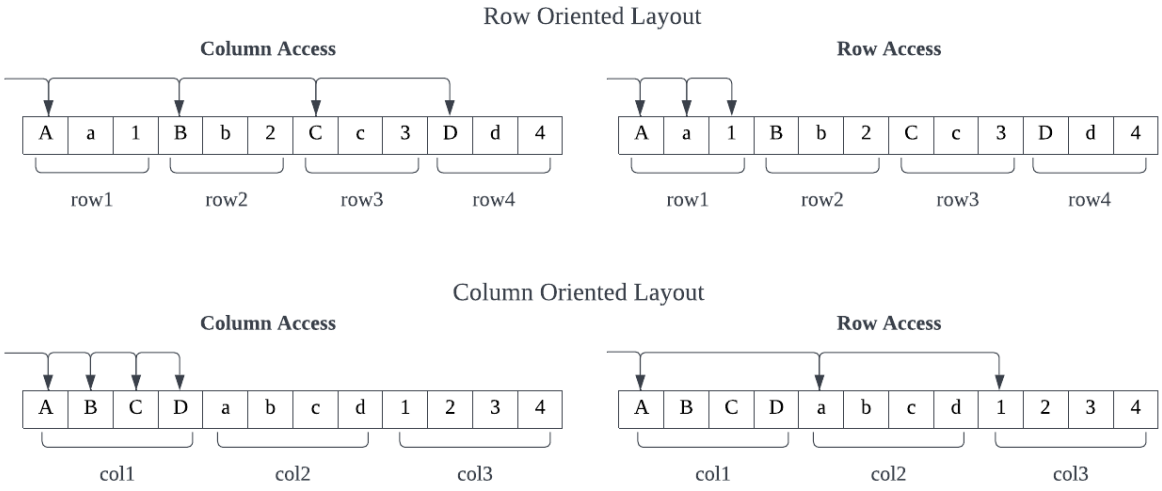

图1：面向列的布局

q编程语言

q 是为 kdb+设计的编程语言。这是一种高度函数式语言，针对富有表现力且高效的数据库管理进行了优化，与 C++、Java 和 Python 等传统通用语言不同 [21]。q 程序在运行时通过脚本解释执行，而非编译。得益于简洁的语法和较小的体积，数据和函数在执行期间可以驻留在内存中，从而实现快速部署和测试。q 中的表是一等实体，像列表和字典等其他数据结构一样存储在内存中。表以有序的列列表实现，每个列表都占用连续的内存空间，以优化顺序访问并利用内存局部性。

q 提供了许多用于数据库操作和管理的专用功能 [23]。q-SQL 是 q 丰富的表查询语法，与 SQL 类似。表可以采用分散存储方式，即沿列垂直拆分，并以独立文件的形式存储在磁盘上。分散存储表还可以进一步分区，进行水平拆分。Python 或 R 可以与 q 结合，用作预处理或可视化工具。由于操作高效、速度快，q 在大数据统计分析和机器学习领域有广泛应用。

kdb+/tick

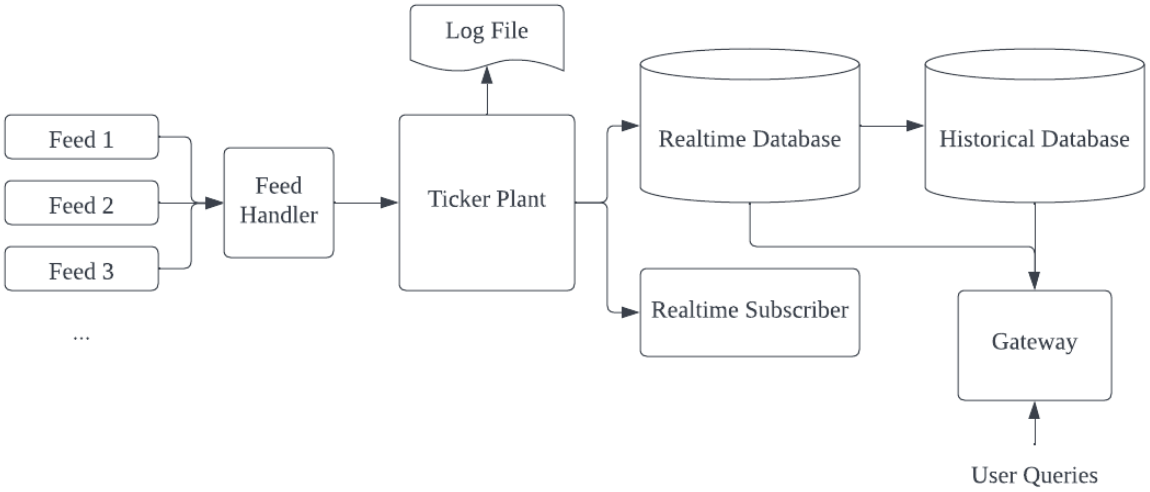

图 2：kdb+/tick 架构

图 2 展示了 kdb+/tick 架构，该架构用于采集、处理和分析实时数据与历史数据 [12]。数据流来自交易场所和彭博等市场数据供应商。馈送处理程序会解析数据，并将其从特定于数据流的格式提取并转换为 kdb + 标准格式。行情数据服务器首次收到消息时，会将数据写入日志文件，以便进行故障恢复。随后，数据会发布给实时订阅者，并保存到内存中的实时数据库。实时订阅者收到数据后，可以对其进行实时处理和分析。每天结束时，内存实时数据库中的日内数据会持久化到磁盘上的历史数据库中；同时，实时数据库也会与日志文件一并清空，以便次日接收新数据。网关用于处理用户对数据库的查询并返回查询结果。结果可能需要来自实时数据库或历史数据库的数据，也可能需要结合两者的数据。

2.1.2 InfluxDB

InfluxDB [24] 是由 InfluxData 开发的开源数据库，针对时间序列数据进行了优化。InfluxDB 用于存储时间序列数据的数据模型和模式设计不同于传统数据库。数据集被水平划分为多个分片，每个分片根据保留策略存储特定时间间隔内的数据。InfluxDB 通过缓存和压缩技术，在写入和读取操作的性能之间取得平衡。为确保可扩展性和可用性，系统做出了一些取舍。传统增删改查中的更新和删除功能受到严格限制，且 InfluxDB 无模式设计不支持跨表连接 [25]。为保证可用性，查询可能返回过时或不完整的结果。

研究表明，InfluxDB 的写入吞吐量是 Creal-time [26], 的 4.5 倍，所需磁盘空间减少 2.1 倍，查询响应时间最高可快 45 倍；与 MongoDB 相比，基准测试结果分别为 2.4 倍、20 倍和 5.7 倍 [27]。CapitalOne 构建了一个高可用、容错的系统，用于实时存储和分析其业务、基础设施及应用指标 [28]。Robinhood 使用 InfluxDB 进行时间序列异常检测，并使用 Faust/Kafka 进行流处理，从而开发了其实时风险监控系统 [29]。

数据模型与模式

与传统数据库相比，InfluxDB 针对时间序列数据优化了其数据模型和模式[30]。图3展示了 InfluxDB 中的数据模型。存储桶位于最高层，用于将具有相同保留策略的数据归为一组。用户可以为不同的存储桶指定不同的保留策略，InfluxDB 会自动删除过期数据。存储桶内的测量值进一步用于描述数据点并对其进行分类。InfluxDB 中的所有数据都必须包含时间戳、标签和字段。标签是一组键值对，通常包含有关数据点且经常被查询的元数据

，因为标签已建立索引，查询速度更快。字段是与时间戳关联的键值对集合，适合用于存储数据值，因为它们没有建立索引。

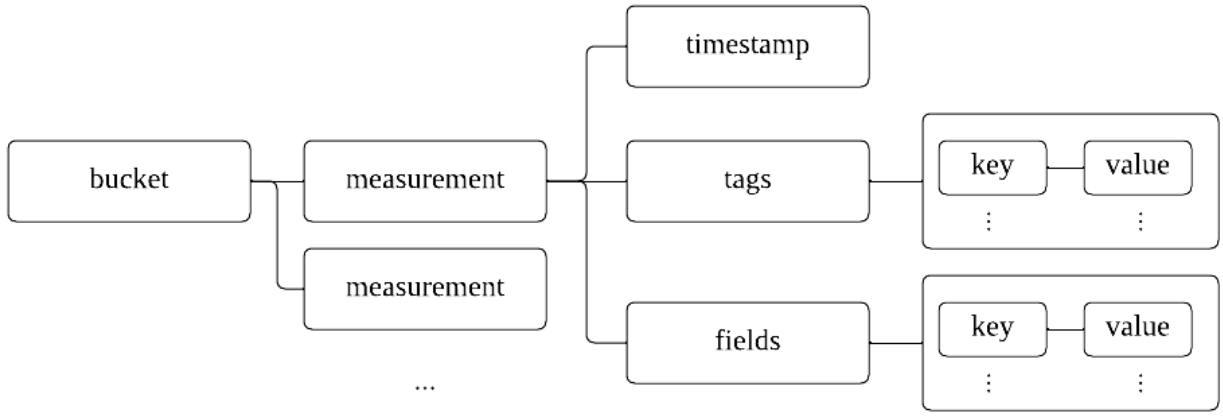

图3：InfluxDB 数据模型

trade table in traditional database

timestamp	symbol	side	amount	price
2022-01-01T00:00:00Z	BTC-USD	buy	0.5	20000.0
2022-01-01T00:10:00Z	BTC-USD	buy	1.0	20010.0
2022-01-01T00:20:00Z	ETH-USD	buy	10.0	2000.0

data points in InfluxDB

		tags		fields	
_time	_measurement	symbol	side	_field	_value
2022-01-01T00:00:00Z	trade	BTC-USD	buy	amount	0.5
2022-01-01T00:00:00Z	trade	BTC-USD	buy	price	20000.0
2022-01-01T00:10:00Z	trade	BTC-USD	buy	amount	1.0
2022-01-01T00:10:00Z	trade	BTC-USD	buy	price	20010.0
2022-01-01T00:20:00Z	trade	ETH-USD	buy	amount	10.0
2022-01-01T00:20:00Z	trade	ETH-USD	buy	price	2000.0

data series in InfluxDB

_measurement	tag set	_field
trade	symbol=BTC-USD,side=buy	amount
trade	symbol=BTC-USD,side=buy	price
trade	symbol=ETH-USD,side=buy	amount
trade	symbol=ETH-USD,side=buy	price

图4：InfluxDB 数据模式

图4以存储加密货币交易的表为例，展示了 InfluxDB 模式与传统数据库模式之间的区别。在 InfluxDB 中，交易表被描述为一种度量。每笔交易的交易符号列和交易方向列被记录为标签，因为它们常用于查询中的筛选器和条件。数量列和价格列描述交易的数据值，并被存入字段。InfluxDB 中的数据序列包含具有相同度量、标签集值和字段键的数据点。如图4所示，示例数据可分为4个序列。InfluxDB 根据时间戳和数据序列对数据进行分区。在同一时间间隔内且属于同一序列的数据点会一起存储在磁盘上的同一分片中。利用时间戳和标签值，可以轻松定位与查询相关的数据，而无需遍历与查询无关的数据。

存储引擎

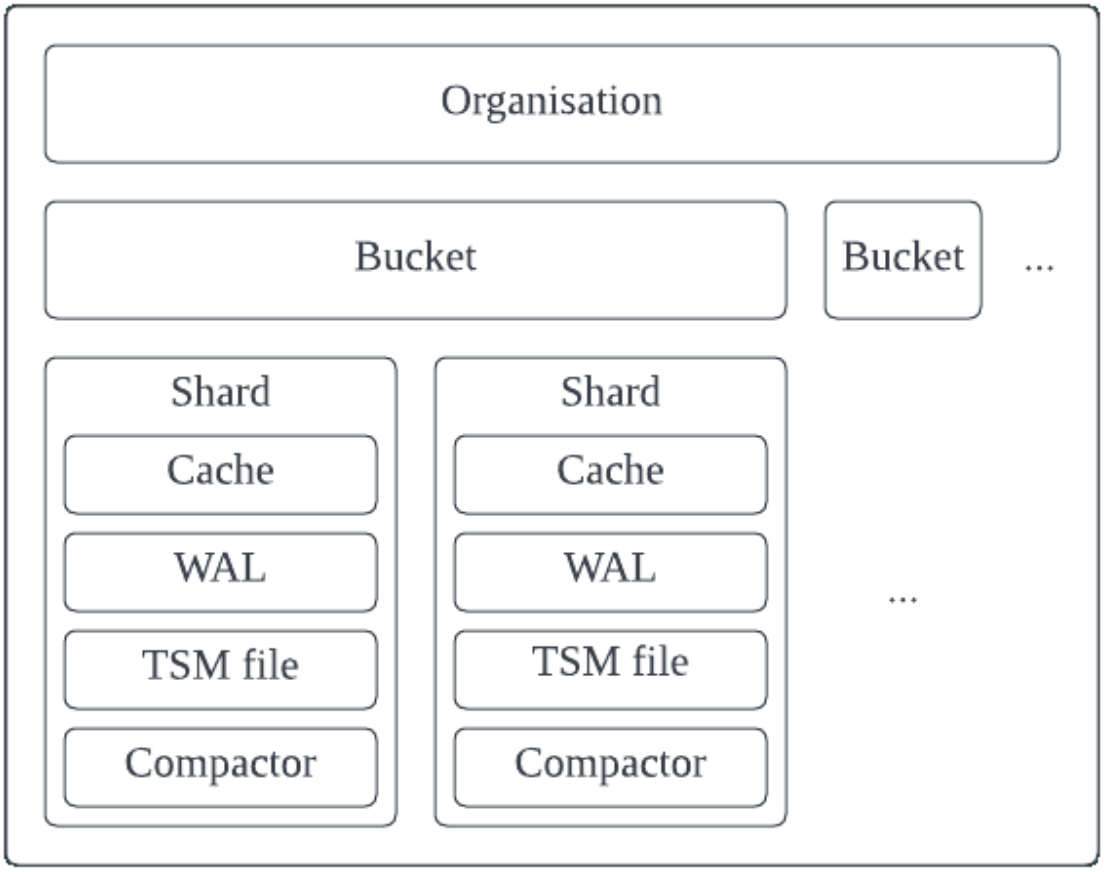

图5：InfluxDB 存储架构

图5展示了 InfluxDB 架构的存储层次结构。数据按不同的保留策略划分为存储桶，并根据时间戳进一步划分为分片。每个分片包含特定时间间隔内的数据序列。为提高写入吞吐量，新数据会写入内存缓存和预写日志（WAL） [31]。缓存和 WAL 中的数据会压缩为磁盘上的只读时间结构合并树（TSM）文件，以优化读取操作。随着 TSM 文件体积增大，压缩器会执行分层压缩，以减少存储占用。

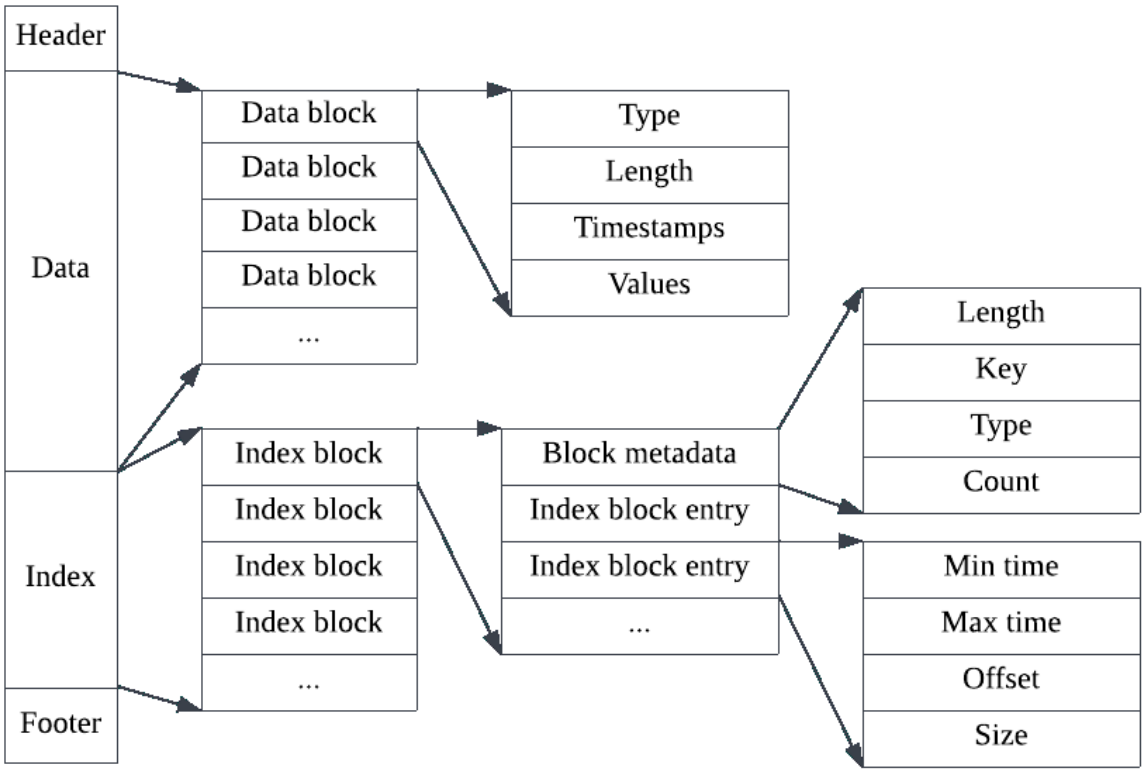

图 6：InfluxDB TSM 文件结构

TSM 由 InfluxDB 开发，旨在优化时间序列数据。图 6 展示了 TSM 文件的结构。TSM 文件包含一个数据段，用于存储数据点的时间戳和字段值；还包含一个索引段，用作数据段的索引。数据段由数据块组成，每个数据块存储一个序列在某个时间间隔内的数据。时间戳采用增量编码进行压缩，字段值则根据数据类型采用不同的压缩方法。执行查询时，数据块是读取的最小单位。索引段中的每个索引块负责存储 TSM 文件中的一个序列。索引块中的元数据包含有关数据序列的信息，其键由度量、标签集和字段键组合而成。后续的索引块条目包含 TSM 文件中每个数据块的位置和时间间隔，这些数据块记录了该序列的数据。文件尾部记录了索引段的偏移量。索引段中的数据块按序列键的字典序排列，索引块条目则按时间排列。这样便可以根据键和时间戳，通过二分查找快速定位数据块。

2.1.3 TimescaleDB

TimescaleDB [32] 是由 Timescale 公司开发的开源时间序列数据库。它是 PostgreSQL 的扩展，针对时间序列数据的存储和处理进行了优化 [33]。作为 PostgreSQL 的扩展，它是一种完全关系型的数据库。与 NoSQL 数据库相比，它通过 SQL 提供强大的分析能力。TimescaleDB 还受益于 PostgreSQL 生态系统，其中包含丰富的扩展、管理工具和可视化仪表板。TimescaleDB 将 PostgreSQL 表转换为超表，并优化查询规划和执行。超表也可轻松部署在分布式环境中，并将数据存储在多个节点上 [34]。TimescaleDB 还包含超函数，这是一组面向时间序列数据分析操作的函数。它们扩展了 SQL，使时间序列查询能够快速执行。

与 PostgreSQL 相比，TimescaleDB 在插入大规模数据时速度快了 20 倍 [35]。TimescaleDB 在基于时间的查询中性能提升了 1.2 至 5 倍，在时间排序查询中的性能提升了 450 倍。

TimescaleDB 还与其他时间序列数据库进行了比较，包括 In- fluxDB [24]。测试表明，TimescaleDB 执行高基数插入的速度快了 3.5 倍，在复杂查询方面的表现则比 InfluxDB 快 3.4 至 71 倍。

超表
超表是 TimescaleDB 实现的核心 [36]。超表是一种时间分区表，每个分区都存储在一个数据块中。每个分区都是标准的 PostgreSQL 表，并带有时间约束，用于指定与该数据块相关的时间间隔。超表是 PostgreSQL 中的父表，而数据块是子表。执行插入或写入查询时，数据块约束可帮助 TimescaleDB 找出需要读取或修改的数据块。由于采用基于时间的分区，TimescaleDB 可以轻松实现水平扩展。

每个数据块都会建立一个局部索引，用于索引其中的数据。由于没有全局索引，TimescaleDB 会将主键与时间戳关联起来，以确保主键唯一。特定时间戳的数据只能存储在一个数据块中，因此数据库只需确保主键值在该局部数据块中唯一。对于近期数据，近期数据块中的数据和索引都存储在内存中。由于时间序列数据通常按时间顺序插入，这有助于快速插入，并加快针对近期数据的查询执行。TimescaleDB 采用基于时间的压缩策略，以适应历史数据的访问模式。较旧的数据块会从行式存储形式转换为列式存储形式，并进行压缩。列式存储形式可以采用特定类型算法来压缩数据。

2.1.4 ClickHouse

ClickHouse [37] 是由 Yandex 开发的面向列的开源数据库系统。ClickHouse 是一种支持扩展版 SQL 的关系型数据库，专为大规模联机分析处理应用场景而构建。ClickHouse 优化的 OLAP 特性包括以大批量方式插入数据、表包含大量列，以及数据插入后很少更新 [38]。此外，ClickHouse 预计查询将在大量行和少量列上执行。如第 2.1.1 节所述，ClickHouse 同样采用的面向列布局适用于 OLAP 工作负载。
ClickHouse 在执行查询时还采用了向量化。向量化每次处理一组行的值，而传统数据库通常一次只处理单行的值。向量化查询执行能够更充分地利用 CPU 缓存 [39]。通过基准测试实验，ClickHouse 的速度已被证明比 Druid 快 24 倍，比 TimescaleDB 快 55 倍 [40]。

2.1.5 MergeTree 与稀疏索引

MergeTree [41] 是 ClickHouse 中的主要存储引擎。将数据插入 ClickHouse 时，数据首先会被划分为多个数据分片。每个数据分片都是按主键排序的一小部分数据，并且各自拥有独立的索引。新数据写入磁盘上的小型数据分片文件后，ClickHouse 会在后台执行合并操作，将它们合并为更大的数据分片。将数据存储在小型数据分片文件中可以实现快速插入，而较大的数据分片文件则有助于提高读取操作的效率。

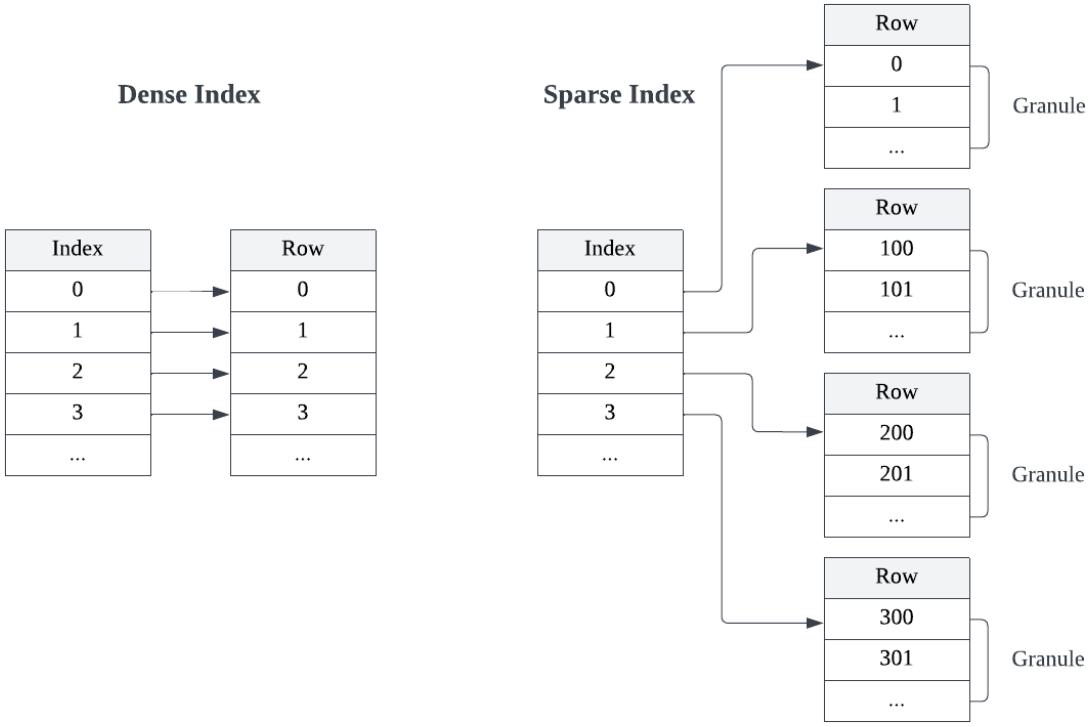

图7：稠密索引和稀疏索引

图7展示了 ClickHouse 用作主索引的稀疏索引。索引充当查找表，用于快速定位某一行，而无需搜索整张表。传统数据库使用稠密索引，这意味着表中的每一行都会在索引中记录一个条目。另一方面，稀疏索引只记录表中的部分行。ClickHouse 使用稀疏索引作为表的主索引 [42]。表中的所有行都被划分为大小相同的粒度块，稀疏索引中的每个条目记录粒度块中第一行的位置。粒度块是 ClickHouse 从磁盘读取数据的最小单位。使用稀疏索引可以显著减小索引的大小，从而使其能够存储在主存中，以便快速访问。稀疏索引在范围查询方面也具有良好的性能。粒度块是 ClickHouse 从磁盘读取的最小数据单位，并且粒度块的大小可以配置。

2.2 数据库基准测试

数据库基准测试是用于评估和比较数据库系统性能的一组实验框架 [43]。为获得可比结果，基准测试根据被测系统及其执行环境进行定义。基准测试的工作负载是对数据库执行的一系列操作，通常根据现实世界用例和访问模式建模。基准测试指标用于量化基准测试的目标，例如吞吐量、延迟和存储。

数据库基准测试具有以下属性 [44]:

- 相关性：工作负载和指标能够模拟典型的系统操作。
- 简洁性：工作负载易于理解。
- 可移植性：该基准测试可轻松在其他系统上复现，而不依赖特定功能。
- 可扩展性：无论系统容量或性能如何，该工作负载都适用。

数据库基准测试可分为三类 [45]:

- 微基准测试：该基准测试评估特定系统组件（包括硬件和软件）或程序功能的性能。

- 端到端基准测试：该基准测试通过模拟现实世界中的典型用例工作负载，评估整个系统的性能。
- 基准测试套件：通过结合微基准测试和端到端基准测试，基准测试套件能够全面评估系统。

基准测试对于评估学术研究并将其应用于工业解决方案至关重要。由于时间序列数据规模庞大且具有独特的访问模式，事务处理性能委员会（TPC）的基准测试并不适用于时间序列数据库，因为这些基准测试主要是为关系型数据库设计并用于关系型数据库的 [1]。针对不同应用场景下的时间序列数据处理，人们开发了多种基准测试。IoTAbench [46] 是为物联网分析场景开发的，其用例是智能计量系统。该基准测试在包含 22.8 万亿条读数、数据量达 727 TB 的数据库上进行了测试，并证明可扩展至其他物联网用例。Linear Road [47] 是一项流处理基准测试，用于模拟交通监控系统，旨在减少交通拥堵和事故。

2.2.1 STAC-M3

STAC-M3 [2] 是证券技术分析中心（STAC）开发的基准测试套件，用于基于时间序列数据进行逐笔金融分析。STAC 基准测试委员会由领先的金融公司和技术提供商组成，致力于为各种金融应用开发并实施标准化基准测试。STAC-M3 中的 Antuco 套件通常称为基线套件，它使用从真实世界的大型数据集中抽取的较小数据集，执行范围广泛的工作负载 [48]。Kanaga 套件是扩展套件，它使用 Antuco 查询的一个子集，并在显著更大的数据集上运行，以评估分布式系统的可扩展性能 [48]。STAC-M3 中的每项查询都会在不同条件下进行测试，其中并发请求数量和数据集大小各不相同。这有助于了解各种解决方案之间的差异，并根据特定用例的需求做出决策。

然而，STAC-M3 并非开源，其规格和所用数据集也无法公开获取。STAC-M3 评估整个系统栈，包括数据库软件和存储硬件。STAC-M3 的测试大多使用 kdb+ 进行，另有少量关于 eXtremeDB 和 Shakti 的报告。

2.2.2 YCSB

雅虎云服务基准测试（YCSB） [3] 是一套面向大规模 NoSQL 应用的开源基准测试套件。YCSB 中的任何基准测试都由客户端、工作负载生成器和核心包组成。客户端连接数据库并执行查询。工作负载生成器旨在支持各种工作负载，以模拟不同的应用场景。核心包负责测试工作负载并记录性能指标。结合可配置的工作负载，YCSB 侧重于使用标准操作而非专用功能来模拟各种用例。因此，YCSB 可轻松应用于各种数据库上的多种用例。

YCSB 已用于比较关系型数据库和 NoSQL 数据库的性能，其中包括 MySQL 和 MongoDB [49]。它还被用于对 Redis、MongoDB 和 Cassandra 等热门 NoSQL 数据库进行比较分析 [50]。作为一个持续发展的开源项目，YCSB 支持 5 种关系型数据库和 40 种 NoSQL 数据库 [51]。YCSB++ [52] 是一组针对 YCSB 的扩展，包含一致性测量和多阶段工作负载，可支持高级优化功能和复杂请求。

3 基准测试设计

本项目旨在设计一套基准测试套件，针对金融数据分析的应用，对时间序列数据库的性能进行评估。首先，定义了需要此类分析的场景以及潜在用例。随后，针对这些用例，考察了相关研究中采用的具体分析方法。基准测试依据现实世界分析中常用的测量值和计算而设计，可作为开展更复杂分析的基础。基准测试还涵盖了频繁执行的数据库操作。本章详细定义了这些用例，并介绍了所设计套件中的各项基准测试。

3.1 用例

本项目定义了这样一种场景：需要数据库系统来存储和查询来自证券交易所、加密货币交易所等市场的数据。数据库存储两类数据集合：交易数据和订单簿数据。订单簿描述了当前针对某种金融工具挂出的所有有效订单。已执行的订单会转化为交易数据中的交易记录。这些数据属于时间序列数据，因为每个数据点都关联着一个时间戳，其值会随时间变化。数据可以收集自交易所交易的各种金融工具，也可以来自多个交易同一种金融工具的交易所。

交易员可以使用此类系统进行实时分析，以辅助决策。研究人员和学者可以通过调查一段时间内的历史数据，深入了解市场。表 1 总结了此类数据库系统在数据写入、数据读取与查询，以及系统容错和效率等类别中的用例。这些类别涵盖了与数据库系统性能相关的不同组件和功能。

类别	**用例**
写入	系统应能够实时解析并汇总来自交易所的数据 实时
	系统应能够批量插入相对大量的 数据
读取和 查询	系统应能够提供实时数据供查询
	系统应能够对日内数据进行相对简单的分析 日内数据
	系统应能够对 大量历史数据进行相对复杂的分析
容错与 效率	系统应能够将数据存储在持久化存储中，并能够 从故障中恢复
	系统应能够以内存高效的方式存储数据

表 1：数据库系统用例分类

3.2 基准测试

为开发基准测试套件，我们研究了使用交易数据和订单簿数据进行数据分析的学术研究。制定这些基准测试的目标是涵盖最常用的分析方式以及数据库系统的各个方面。表2列出了每项基准测试的规格。每项基准测试都根据其内容分配了一个标识符，以便于查阅。这些基准测试采用查询的形式设计，用于模拟系统用户实际执行的分析型查询。

类别	ID	规格	I/O 强度	计算机-度 强度
交易	T-V1	1分钟间隔内的平均交易量 一天内	轻量级读取	轻量级
	T-V2	一个月内的日均交易量	重型 read	轻量级
	T-VWAP	成交量加权平均价格（VWAP），历时 一天内的1分钟间隔	轻量级读取	重型
订单 Book	O-T	在随机时间生成最优买卖报价	轻量级读取	轻量级
	O-B1	查找一周内的最高买价	轻量级读取	轻量级
	O-B2	查找一个月内的最高买价	重型 read	轻量级
	O-S	一天内的买卖价差	轻量级读取	重型
	O-V1	使用以下方法查找顶层市场深度 以 1分钟间隔计算平均成交量 超过一周	轻量级读取	重型
	O-V2	使用以下方式查找第5档的市场深度 使用1小时区间的平均成交量，时间跨度为 一个月	重型 read	重型
	O-NBBO	某一天的全国最佳买价和卖价（NBBO）	轻量级读取	轻量级
复杂查询	C-R	一天内以5分钟间隔计算的中间报价收益率	轻量级读取	轻量级
	C-VT	将波动率计算为每小时标准 5分钟间隔内成交价格收益的偏差 一天的间隔	轻量级读取	重型
	C-VO1	按小时计算波动率标准 5 分钟间隔内中间报价收益率的偏差 隔，持续一天	轻量级读取	重型
	C-VO2	每 4 小时计算一次标准差，以此作为波动率。 1小时内中间报价收益率的标准差 一周的区间	轻量级读取	重型
写入	W	将一天的交易和订单簿数据写入 持久化存储	重型 写入	轻量级
存储效率	SE	比较同一数据集在数据库中的磁盘存储空间 与原始数据文件中的磁盘存储空间	-	-

表2：基准测试规范

这些基准测试分为5个类别。交易类别包括对交易数据进行的业界常用计算，例如平均交易量和成交量加权平均价格（VWAP）。此外，这些计算相对简单，无需执行包含连接的复杂查询。第二个类别涵盖对订单簿数据进行的常见操作和分析，例如买卖价差和全国最佳买价和卖价（NBBO）。需要相对复杂的连接和嵌套查询的基准测试归入复杂查询类别。这些查询所执行的计算（例如中间报价收益率）被研究人员广泛使用，通常也是更复杂的定制分析的基础。标识符为 W 的基准测试旨在评估数据库系统的批量写入能力。此外，还会评估数据库的存储效率。

每项基准测试还会根据其输入/输出（I/O）强度和计算强度进行分类。输入/输出强度描述执行期间需要读取或写入的数据量。当查询所需访问的数据分布在大量数据文件或数据块中时，输入/输出强度较高。计算强度根据执行查询时对数据进行的计算量对查询进行分类。计算强度较高的查询不一定需要大型数据集，但会使用数据进行复杂计算。各项基准测试在输入/输出强度和计算强度方面的分类各不相同，以涵盖数据库系统的不同方面。

3.2.1 交易

交易量

$$AverageVolume = \sum_{t \in T} t.amount$$
$$where\ T = \{t \mid t.timestamp \in t_i, t.side = s_i, t.symbol = sym_i\} \tag{1}$$

方程（1）展示了计算平均成交量的公式。 t_i 是时间间隔集合中的一个元素。s_i 来自 $\{buy, sell\}$ ，用于表示该交易是在买方还是卖方执行的。 sym_i 是数据集中记录的、正在交易的金融工具的交易品种。平均成交量的计算方法是：对在特定时间间隔内、由一方执行且交易特定交易品种的所有订单的交易数量求和。交易量一直被视为市场流动性分析的传统衡量指标 [53]。研究发现，平均交易量与股票回报密切相关，并且能够预测未来收益率，因此可用于制定交易策略 [54]。

为实现基准测试 T-V1， t_i 是特定一天内 1 分钟间隔集合中的一个元素。 sym_i 是这一天记录的所有交易品种。通过将 t_i 设为 1 个月内的 1 小时间隔来查询 T-V2。

交易价格

$$VWAP = \frac{\sum_{t \in T}(t.amount \times t.price)}{\sum_{t \in T} t.amount}$$
$$where\ T = \{t \mid t.timestamp \in t_i, t.symbol = sym\} \tag{2}$$

成交量加权平均价格（VWAP）通过方程（2）计算。 $t.amount \times t.price$ 表示交易资产的总价值，而 VWAP 的计算方式是用总交易额除以总交易量。VWAP 根据交易量进行了调整，因此比单纯的交易价格能提供更多见解。VWAP 可用作基准价格，以评估交易执行对市场的冲击 [55]。可以制定交易策略来优化 VWAP。要计算基准测试 T-VWAP， t_i 是特定一天内一组 1 分钟间隔中的每个元素。 sym 是计算所针对的特定交易资产的交易符号。

3.2.2 订单簿

Buy	Price	Sell
	104	2500
	103	3000
	102	2000
	101	1000
500	100	
1000	99	
2500	98	
1500	97	

Ask / Offer:
Orders to sell

Bid:
Orders to buy

图 8：订单簿示例

图 8 展示了订单簿的一个示例。订单簿是按金融工具列出的、已提交的有效买单和卖单的有序列表。卖方一侧通常称为卖价或要约价，而买方一侧称为买价。订单簿按价格档位排列，即多个订单希望买入或卖出的价格。每笔订单都包含买方或卖方的信息、数量以及其希望执行的价格。例如，图 8 的第一行显示，所有价格为 104 的卖单合计为 2500 单位。在电子交易所中，撮合引擎用于撮合同一价格档位的买单和卖单，以执行交易。已执行的订单会转化为交易，并从订单簿中移除，因此卖价和买价两侧不会交叉。

最优买卖报价

最优买卖报价是指订单簿中的最高买价和最低卖价。在图 8 的示例中，最优买卖报价对应价格档位 100 和 101。最优买卖报价体现了订单成交的要求，也会影响下单策略以及订单提交、撤单和改单等操作 [56]。基准测试 O-T 会在数据集中随机选择一个时间戳，并查询特定交易符号当前的最优买卖报价。这项操作预计需要在日内交易时段快速完成，因为它所需的磁盘读取和计算量都相对很少。

订单价格

与最优买卖报价相关，O-B1 和 O-B2 用于查找特定时间间隔内的最高买价。这项操作可用于分析历史数据，或将当前市场数据与历史数据进行比较。O-B1 和 O-B2 的计算强度较低，因为它只是对一组数据进行简单的聚合运算，以找出其中的最高值。这两项基准测试所处理的时间间隔不同，反映了需要读取的数据量。它们旨在评估数据库系统在数据量方面的可扩展性。

$$Bid - Ask\ Spread = MIN(ask) - MAX(bid) \tag{3}$$

方程（3）表明，买卖价差的计算方法是用最低卖价减去最高买价。买卖价差是交易的最低风险或成本。它常被用作市场流动性的指标，因为流动性充足的资产通常价差较小。它也用于分析证券的日内行为 [57], , 这是基准测试 O-S 模拟的使用场景。

全国最佳买价和卖价（NBBO）是指在多个交易所中计算得出的某项金融工具的最高买价和最低卖价。NBBO 通过找出不同交易所中的最优价格，使交易员受益。基准测试 O-NBBO 专为日内分析而设计，并旨在将数据集从单一交易所扩展至多个交易所，以增加数据集的规模。

订单量

$$\begin{gathered} Market\ Depth = \sum_{o \in O} o.amount \\ where\ O = \{o \mid o.price \in \{l_1, l_2 ... l_n\}, o.side = s_i, o.symbol = sym\} \end{gathered} \tag{4}$$

订单簿中包含每笔订单所愿意交易的资产数量。方程（4）展示了如何使用订单量计算 n 价格档位的市场深度。在任意时刻，市场深度是买入或卖出方向上所有价格不超过指定价格档位的订单的累计订单量。例如，卖方 n 价格档位的市场深度，是所有价格低于 n档位价格的价格档位上的订单量之和。市场深度表示需要交易多少资产才能使价格达到 n价格档位。市场深度是市场流动性的指标。市场深度越大，市场流动性越高，这意味着执行交易时资产价格很可能保持稳定。研究发现，市场深度会影响价格发现；根据市场深度计算出的成交量失衡，与未来的短期收益相关 [58]。

要计算给定时间段的市场深度，需要将该时间段划分为多个时间间隔，每个时间间隔的订单量为该时间间隔内所有订单簿快照的平均订单量。基准 O-V1 和 O-V2 分别采用 1 周和 1 个月的时间段，以及 1 分钟和 1 小时的时间间隔。

3.2.3 复杂查询

$$Return = log(price_t) - log(price_{t-1}) \tag{5}$$

$$Execution\ Price\ Return = Return\ where\ price = t.price \tag{6}$$

$$Mid\ Quote\ Return = Return\ where\ price = \frac{MIN(o.ask) - MAX(o.bid)}{2} \tag{7}$$

资产在时间 t 的收益率，按时间 t 和时间 $t-1$ 之间价格的对数差计算，如方程 (5) 所示。方程 (6) 中的成交价格收益率使用时间 t 和 $t-1$ 的成交交易价格计算。方程 (7) 使用中间报价价格（即最低卖价与最高买价的平均值）来计算中间报价收益率。要计算给定时间段内的收益率，需将该时间段划分为多个时间间隔，并将每个时间间隔内时间戳最新的价格数据作为收盘价提取到列表中。随后，对列表中的每个元素应用方程 (5) 来计算收益率。

$$\begin{gathered} Volatility = Standard\ Deviation(R) \\ where\ R = \{return | return.timestamp \in t_i\} \end{gathered} \tag{8}$$

方程 (8) 展示了如何使用收益率分析市场波动率。波动率通过某一时间段内收益率的标准差计算得出。波动率指资产价格变化的不确定程度。波动率越高，价格变化就可能越剧烈或越难以预测，而非稳定上涨或下跌。波动率可帮助交易员判断短期风险 [53]。

基准测试 C-R 以 5 分钟间隔计算中间报价收益率，用于日内分析。基准测试 C-VT 和 C-VO1 使用每小时累计的 5 分钟价格收益率计算波动率。基准测试 C-VO2 使用粒度较低的数据计算较长时间段内的波动率，旨在模拟历史数据分析。

```
1 SELECT time_bucket ( '1 小时 ', timestamp ) AS one_hour , 2 stddev ( return ) AS volatility 3 FROM 4
( 5
ret
FR
TI
bo
on
```

列表 1：在 PostgreSQL 中实现 C-VO1

复杂查询类别中的基准测试旨在评估数据库系统执行复杂查询的能力，包括嵌套查询和连接。清单 1 展示了在 PostgreSQL 中实现 C-VO1 的示例，其中包含两个嵌套查询和一个内连接。该查询使用中间报价收益率计算一天的波动率。最内层查询找出每个 5 分钟间隔内收盘价记录的时间戳。随后将这些时间戳与整个订单簿表连接，生成包含这些记录的表。然后，将中间报价收益率计算为相邻行中间报价价格对数的差

值。LAG 函数用于访问前一行。最外层查询将 1 小时间隔内收益率的标准差作为波动率计算。

3.2.4 写入

基准测试 W 用于衡量数据库系统的批量插入性能。数据通过两种通用文件格式（例如 .csv）的文件插入数据库。将存储单日交易和订单簿数据的文件分别载入持久化存储。本基准测试记录从开始加载和解析文件，到数据写入数据库存储并可供查询所需的时间。若数据库系统需要，文件中的数据模式也会一并提供。

3.2.5 存储效率

$$Storage\ Efficiency = \frac{Size\ of\ data\ in\ database\ storage}{Size\ of\ data\ on\ file} \times 100\% \tag{9}$$

基准测试 SE 使用方程 (9) 以百分比表示存储效率。数据库系统通常采用压缩和压实技术来减小数据库的磁盘存储空间。然而，压缩和压实程度越高，执行查询时解压数据所需的时间也越长。存储效率与读取性能之间存在权衡。许多数据库系统会通过建立索引来提升查询性能，以便快速定位数据。另一方面，索引会占用实际数据值之外的额外磁盘空间，而且索引文件的大小会随数据库规模增大。

3.3 指标

基准测试中测量的主要指标是查询延迟。查询延迟是指从向服务器发送查询到收到查询结果所经过的时间。查询延迟包含两部分：通过网络发送数据所花费的时间，以及服务器端的查询执行时间。然而，网络延迟无法反映数据库性能，并且受网络带宽和物理距离等因素影响。因此，假设用户与服务器之间的网络连接可靠，可以忽略网络延迟。忽略网络延迟后，便可使用各数据库系统的内置计时器，将查询延迟作为查询执行时间。

次要指标是查询执行期间使用的存储空间总量。虽然高效利用存储对数据库而言十分理想，但响应客户端查询时，查询延迟更为重要。随着存储设备价格不断降低，存储限制可以通过增加横向扩展存储轻松克服。

4 数据收集

本研究以从加密货币交易所收集的数据为例，展示第 3.1 节所定义的真实用例中收集的数据。用于实现基准测试套件的数据包括两类：通过数据收集管道实时收集加密货币交易所的真实交易数据；此外，还会生成随机数据，以模拟真实世界的数据。本章将详细介绍数据收集过程，并展示数据集。

加密货币交易所是人们可以交易加密货币和数字货币等资产的在线平台。它们在市场中发挥着与传统证券交易所相同的作用。由于加密货币交易所采用区块链技术实现，因此在这些交易所上进行的活动对网络中的每个人都是可见的。用户可以轻松访问实时交易数据和历史数据。由于区块链具有分布式特性，这些数据也很可靠，因为它们已由矿工进行验证。加密货币交易所中的交易与现实世界交易所中的交易完全相同，区别仅在于所使用的货币。根据具体交易所的不同，加密货币交易所可能拥有类似于现实世界交易所的大规模数据流。这些原因使加密货币交易所数据非常适合用于模拟现实世界金融应用的数据和工作负载。

4.1 数据收集流程

本项目使用 Cryptofeed [59] 从加密货币交易所收集数据。它是一个开源 Python 库，专为处理来自加密货币交易所的数据流而设计。它可用于获取

实时数据或历史数据。Cryptofeed 会将任意组合的受支持交易所的数据流合并为一个标准化且规范化的数据流。开发人员可以在不同的数据流通道上注册回调函数，包括交易、行情更新和订单簿更新，以处理数据。

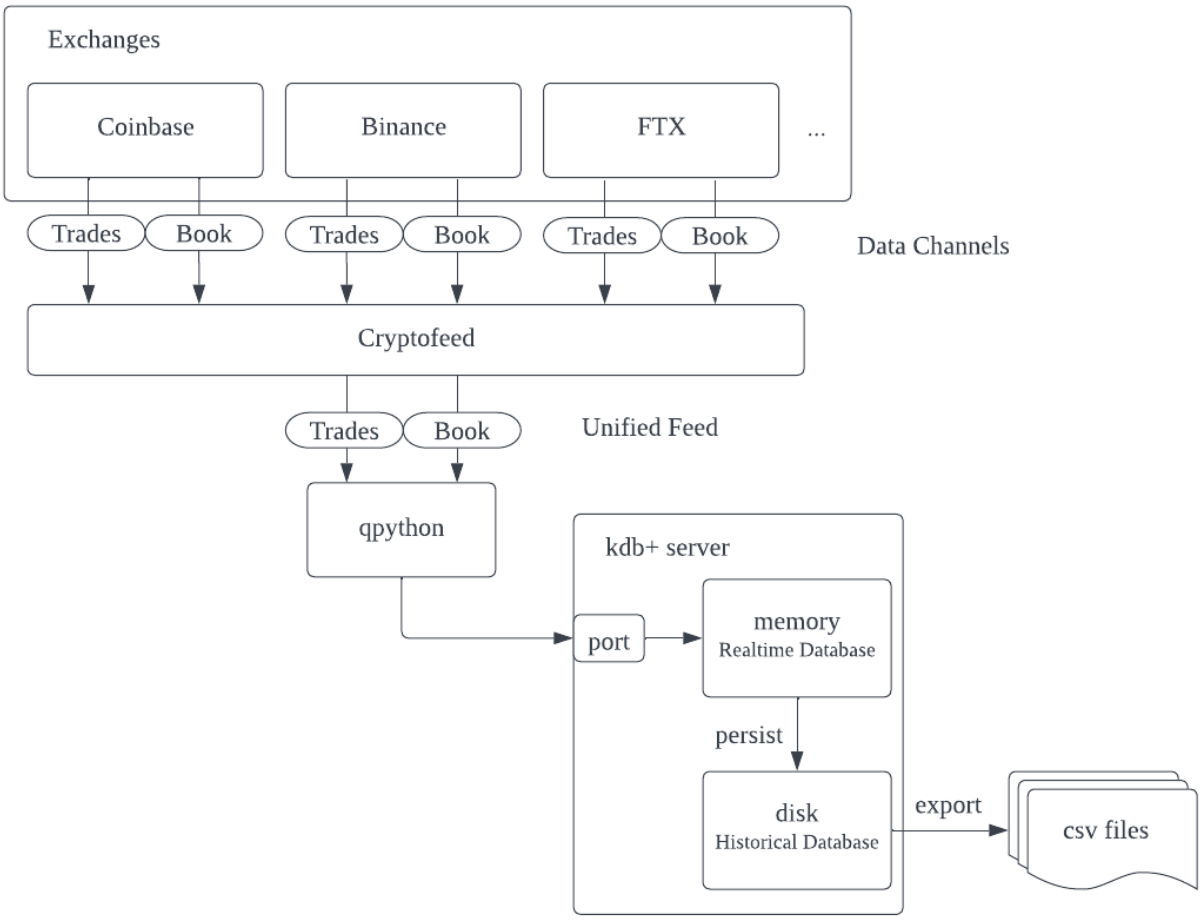

图9：Kdb+ 数据收集管道

图9展示了使用 kdb+ 的数据收集管道。多个交易所会根据各自的操作生成数据流。Cryptofeed 中定义了一个 FeedHandler，用于订阅指定交易所的数据流，并将它们合并为一个标准化数据流，同时将数据转换为 Python对象。每个交易所都可以为不同类型的数据流打开多个数据通道。本项目订阅交易和订单簿通道以收集数据，而 Cryptofeed 会为每个通道生成统一的数据流。

Cryptofeed 生成的统一数据流的回调函数使用 qPython 实现。qPython [60]是一个 Python 库，可实现 Python 与 kdb+/q进程之间的通信。qPython 首先与 kdb+ 服务器建立连接，该服务器上有一个 q进程在特定端口监听。收到订阅数据流的数据后，Cryptofeed 会调用回调函数。回调函数会对数据进行预处理、构建查询，并将数据发送到 kdb+ 服务器进行存储。每个通道都实现了一个 qPython 回调函数。

当 kdb+ 服务器接收到开放端口上的数据时，会先将数据保存在内存中的实时数据库中。每天结束时，实时数据库中的内容会持久化到磁盘上的历史数据库中。因此，实时数据库可以腾出空间来存储新接收的数据。本项目在本地机器上实现 kdb+ 服务器，并将交易和订单簿频道数据分别存储在两个

表中。历史数据库会导出为 CSV 文件，作为导入其他用于测试的数据库系统的原始数据文件。每天都会为每个表生成一个单独的文件。

4.2 随机数据生成

随机生成一天的交易和订单簿数据，用于实施标识符为 W 的写入基准测试。数据使用 kdb+ 生成，并分别保存到两个 CSV 文件中。交易数据生成 1,000,000 行，订单簿更新数据生成 1,500,000 行，其格式与采集的真实数据相同。生成的数据包含相同的交易品种，每条记录的价格和数量均根据采集的现实世界数据中相应交易品种的平均值随机生成。

4.3 数据集

作为现实世界用例中加密货币交易所数据流的样本，本项目采集了 3 种交易品种的数据，即 BTC-USD、ETH-USD 和 USDT-USD。BTC、ETH 和 USDT 的实时交易存储在交易表中。订单簿表订阅二级订单簿数据，其中包含每个价格档位上的买价和卖价数量。由于最顶部的 20 个价格档位数据与资产价格关系更密切，也更有可能撮合成交，因此只采集这些数据。订单簿表存储从交易所接收的订单簿更新的 $1\ 40$ 样本。这是因为订单簿更新发生得非常频繁，如果记录所有更新，就会产生大量冗余数据。

该数据集可分为三个日期区间：2022-06-01、2022-06-18 至 2022-07-17，以及 2022-07-20。2022-06-01 的数据是按照第 4.2 节所述随机生成的。2022-06-18 至 2022-07-17 的数据仅从 Coinbase 交易所收集，用于实现大多数基准测试。2022-07-20 收集的数据仅包含来自多个交易所的订单簿数据，包括 Coinbase、BinanceUS、FTX、FTXUS 和 Bitstamp，用于实现基准测试 O-NBBO。
该数据集的交易表约有 2000 万条记录，订单簿表约有 3300 万条记录。导出的 CSV 数据文件总大小为 25.14 GB。表3按交易符号和表展示了有关该数据集的信息。BTC-USD 和 ETH-USD 的交易数量相近，且订单簿更新数量都很大。在多个交易所进行交易时，它们的交易频率大致翻了一倍。USDT-USD 的交易和订单簿更新数量明显少于另外两个记录的交易品种。

Date	交易符号	表	行数（百万）	频率（秒）
2022-06-18 - 2022-07-17	BTC-USD	交易	8.81	6.86
		订单簿	15.66	10.98
	ETH-USD	交易	10.01	8.41
		订单簿	14.06	10.01
	USDT-USD	交易	0.76	1.80
		订单簿	0.92	1.65
2022-07-20	BTC-USD	订单簿	1.43	16.79
	ETH-USD	订单簿	1.22	14.42
	USDT-USD	订单簿	0.11	1.89

表3：数据集信息

5 实现

基准测试套件通过编写与每项基准测试相对应的查询，并在不同数据库系统上执行这些查询来实现。目前已针对 4 种数据库系统完成实现，包括 kdb+、InfluxDB、TimescaleDB 和 ClickHouse。此外，kdb+同时支持内存中和磁盘模式。这些数据库均安装在用于测试的本地机器上。本章详细介绍实现基准测试过程中的 3 个步骤：数据集导入、基准测试实现和基准测试执行。

选择 kdb+、InfluxDB、TimescaleDB 和 ClickHouse 这 4 个数据库，是因为它们各有特点，并针对时间序列数据进行了优化。kdb+ 在基准测试中表现出色，并且

在金融行业得到广泛应用。kdb+ 采用列式布局在磁盘上存储数据，并配有以专用语言 q 编写的轻量级查询引擎。kdb+ /tick 架构提供内存模式的实时数据库（RDB）和磁盘模式的历史数据库（HDB）。InfluxDB 是一款通用型时间序列数据库。其设计了专属的数据模型和存储文件，以优化时间序列数据处理。TimescaleDB 是 PostgreSQL 的扩展，这意味着它是完全支持 SQL 的关系型数据库。ClickHouse 专为大数据联机分析处理（OLAP）而设计，能够处理大量数据和复杂的分析型查询，这正是该用例所需的。

5.0.1 数据集导入

从 CSV 文件将数据集导入各个数据库系统。所有数据库中的数据库模式和数据类型均保持一致。例如，kdb+ 允许将列表插入表格单元格，而其他数据库只允许在表格单元格中存储单个元素。为保持一致性，kdb+数据库采用与其他数据库相同的数据类型，不使用这一功能。对于数据模型与其他数据库不同的 InfluxDB，数据集模式按照 InfluxDB 推荐的方式实现。类别值列通常用于筛选查询中的数据，因此会作为标签导入；连续数值列通常用于计算，因此存储为字段。此外，所有数据库的分区规则均设为 1 天。

5.0.2 基准测试实现

为实现基准测试套件，将各项基准测试转换为可在这 4 个数据库中执行的查询。每项基准测试对应一个查询。一项基准测试可分为两部分：数据和操作。操作（如数值计算或聚合）作用于从数据集中选取的数据。表4展示了各项基准测试与其日期范围之间的对应关系。

日期范围	表	基准测试编号
2022-06-18	交易	T-V1, T-VWAP , C-VT
	订单簿	O-S, C-R, C-VO1
2022-06-18 - 2022-06-24	交易	
	订单簿	O-B1, O-V1, C-VO2
2022-06-18 - 2022-07-17	交易	T-V2
	订单簿	O-T, O-B2, O-V2
2022-07-20	订单簿	O-NBBO

表4：数据范围及对应的基准测试

每种数据库的实现都使用该数据库专属的查询语言，例如 kdb+ 使用 q，InfluxDB 使用 Flux。TimescaleDB 和 ClickHouse 均使用 SQL 作为查询语言，并针对聚合、算术计算和分组等领域扩展了 SQL，加入了各自的函数。优先使用这些函数，而非原生 SQL 实现，因为数据库系统已对这些函数进行优化，以提升性能。此外，还确保不同数据库针对同一基准测试返回的结果保持一致：结果应具有相同的列，且行的顺序也应相同。这可能需要执行基准测试中未指定的额外操作，例如对 ClickHouse 的结果进行排序，因为其结果不像其他数据库那样有序；或者对 InfluxDB 的列进行透视，因为其数据模型有所不同。

根据第 2.1.1 节所述的 kdb+ /tick 架构，kdb+ 数据库分为内存模式和磁盘模式。由于内存中的实时数据库仅存储日内数据，内存模式的实现仅包括针对单天数据执行的基准测试。所有基准测试均在磁盘上的 kdb+ 数据库中实现。

5.0.3 基准测试执行

基准测试执行过程由 Python 自动完成。Python 通过特定端口连接到托管在本地机器上的数据库服务器。查询会发送到数据库并执行。查询结果会以 Python对象的形式返回。用于执行基准测试套件的机器是一台个人电脑，配备英特尔 i7-1065G7 处理器和 16GB 内存。所有数据库都将数据存储在固态硬盘中。

执行期间会提供足够的内存空间，并且同一时间只测试一个数据库。每项基准测试的结果通过对 10 次执行的结果取平均值获得。

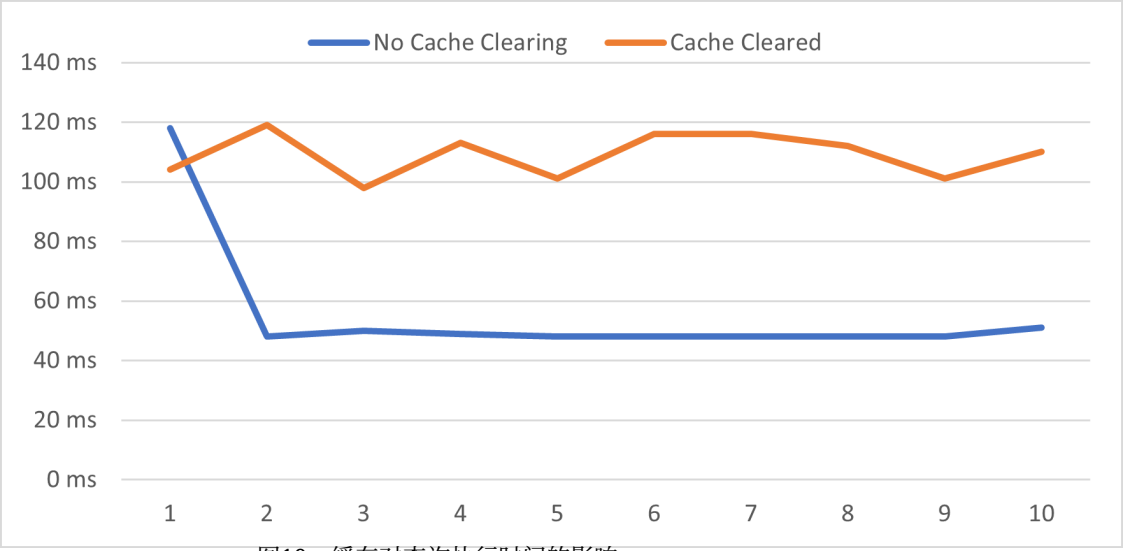

图10：缓存对查询执行时间的影响

执行期间，操作系统会将从磁盘读取的数据存入缓存。如果再次执行相同或类似的查询，由于数据是从缓存而非磁盘读取，查询执行速度会更快。由于数据库系统将数据存储在磁盘上，这会带来一个问题：缓存会影响查询执行时间，并可能导致性能评估失真。图10展示了缓存效应如何影响执行时间。如果不清除缓存，首次执行后，执行时间会大幅缩短。如果在每次执行前清除缓存，查询执行时间会随磁盘访问时间而波动，但总体保持稳定。为避免此类问题，每次执行查询前都会清除备用列表缓存。

6 结果与评估

通过在 kdb+ 内存中、kdb+ 磁盘上、InfluxDB、TimescaleDB 和 ClickHouse 中执行基准测试，获得了实验结果。本章将展示这些结果。此外，还从数据摄取与存储、读取查询、计算密集型查询和复杂查询等方面分析了受测数据库系统的性能。最后，本文还将评估所开发的基准测试套件，并与相关工作进行比较。

6.1 数据摄取与存储

数据摄取和存储性能通过基准测试编号 W 和 SE 进行评估。由于数据库保存在主存中而非持久性内存中，因此 kdb+ 内存模式未实现这两项基准测试。

表 5 展示了各数据库的基准测试结果。基准测试 W 显示，kdb+ 和 TimescaleDB 的数据导入速度明显快于 InfluxDB 和 ClickHouse。与 4 个数据库中速度最快的 kdb+ 相比，InfluxDB 慢 9.6 倍，ClickHouse 慢 22.6 倍。图 11 展示了根据基准测试结果计算的数据摄取吞吐量对比。kdb+ 能够以 47.35 MB/s 的速度从 CSV 文件导入数据，吞吐量是 TimescaleDB 的 1.6 倍。表 5 中基准测试 SE 的结果显示，InfluxDB 和 kdb+ 的存储效率优于 TimescaleDB 和 ClickHouse。InfluxDB 的压缩率最高，达到 83.73%，比 kdb+ 高 6.5%，比 TimescaleDB 高 10.95%。ClickHouse 的压缩机制表现相对较差，其压缩率接近 100%。

基准测试 ID	kdb+ 磁盘上	InfluxDB	TimescaleDB	ClickHouse
	查询延迟（毫秒）			
W	33,889	324,854	53,150	765,000
	压缩率（%）			
SE	90.20	83.73	94.68	99.65

表 5：数据加载性能

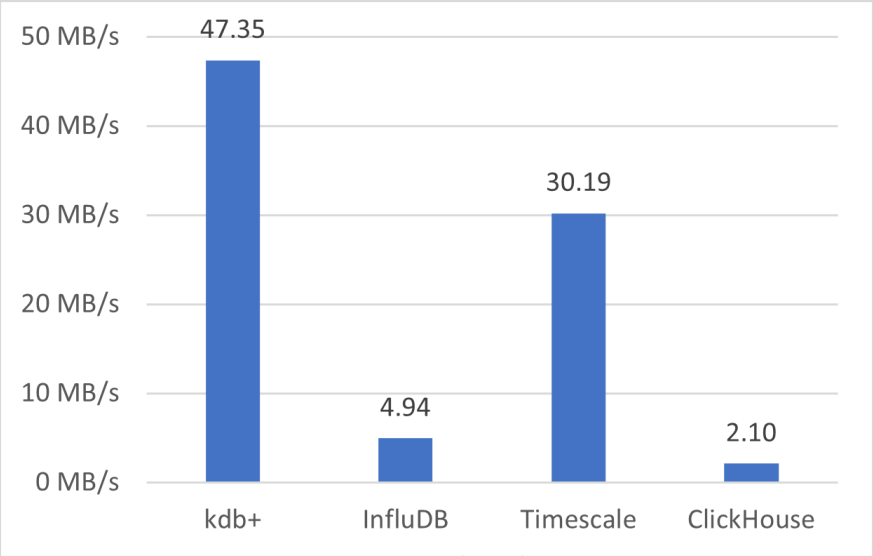

图 11：数据摄取吞吐量（MB/s）

InfluxDB 数据摄取缓慢，可能是由于其 TSM 文件结构和压缩方法所致。InfluxDB 的数据模式要求先将数据分类为序列，再存储到 TSM 文件中。此外，InfluxDB 会在将数据存入 TSM 文件之前进行数据压缩；由于其压缩率最高，这一过程可能比其他数据库更加耗时。InfluxDB 会将每个 TSM 文件的索引构建在文件内部。由于 TSM 文件的大小最高可达 2GB，构建索引可能会很耗时。
对于本项目的数据，ClickHouse 在批量写入和存储效率方面表现最差。这可能是因为 ClickHouse 的设计目标是处理大规模联机分析处理工作负载，而非专门处理时间序列数据。这意味着 ClickHouse 不会针对按时间戳对数据进行分区和排序进行优化。此外，ClickHouse 不使用时间序列数据库常用的压缩技术，例如对时间戳进行增量压缩。

结论

- kdb+ 在本节中整体性能最佳，同时也是数据摄取表现最好的。
- InfluxDB 的存储效率最佳。
- ClickHouse 在本节中的性能最差。

6.2 读取查询

基准测试 T-V1、T-V2、O-T、O-B1、O-B2 和 O-NBBO 被归类为主要包含读取操作且计算强度较低的查询。表6展示了这些基准测试的结果。在仅包含一天数据的基准测试中，kdb+ 内存模式的性能最佳。对于基准测试 O-NBBO，kdb+ 内存模式的速度比磁盘模式快 7 倍。这是预期之中的，因为采用内存模式能显著缩短读取数据所花费的时间。在磁盘数据库中，kdb+ 的整体性能最佳，在大多数查询中具有最低的查询延迟。InfluxDB 的整体性能位居第二。ClickHouse 通常比 InfluxDB 慢，但比 TimescaleDB 快。

在 T-V2 中，InfluxDB 的速度比 kdb+ 快 2.7 倍，但在 T-V1 中速度相近。T-V1 与 T-V2 的区别在于输入数据和聚合窗口。T-V1 需要 1 天的数据，而 T-V2 需要 1 个月的数据。T-V1 中聚合操作的时间窗口为 1 分钟，T-V2 中则为 1 天。由于 InfluxDB 在 T-V2 中明显更快，这可能表明 InfluxDB 在针对时间窗口内数据进行聚合的窗口函数方面优化得更好。

基准测试 ID	kdb+ 内存中	kdb+ 磁盘上	InfluxDB	TimescaleDB	ClickHouse
	查询延迟（毫秒）				
T-V1	57	81	93	469	272
T-V2	-	740	273	3,700	1,991
O-T	5	11	124	10	706
O-B1	-	94	146	3,938	962
O-B2	-	287	440	12,847	3,202
O-NBBO	5	34	191	1,614	454

表6：读取查询的性能

6.2.1 结论

- 读取查询性能：kdb+ > InfluxDB > ClickHouse > TimescaleDB。
- 与 kdb+ 相比，InfluxDB 对窗口聚合函数的优化可能更好。

6.3 计算密集型查询

通过 T-VWAP、O-S、O-V1 和 O-V2 基准测试评估数据库在计算密集型查询方面的性能。表7展示了这一类别的实验结果。kdb+ 的内存中模式和磁盘模式在所有数据库中速度最快。kdb+ 的内存中模式与磁盘模式在 O-S 中的表现差异很小。这是因为大部分查询延迟与计算时间有关，而非数据读取时间。InfluxDB 在 O-V1 和 O-V2 中表现良好，而 TimescaleDB 和 ClickHouse 则明显较慢。在 T-VWAP、O-V1 和 O-V2 中，ClickHouse 的查询延迟低于 TimescaleDB；但在 O-S 中则较慢。

基准测试 ID	kdb+ 内存中	kdb+ 磁盘上	InfluxDB	TimescaleDB	ClickHouse
	查询延迟（毫秒）				
T-VWAP	26	75	12,716	334	202
O-S	1,352	1,386	623,732	2,182	14,056
O-V1	-	533	373	4,699	1,626
O-V2	-	4,375	4,420	17,092	10,180

表7：计算密集型查询的性能

T-VWAP 和 O-S 中 InfluxDB 的查询执行时间显著偏长，这可能是由 InfluxDB 查询语言和数据模型的局限性造成的。由于 InfluxDB 没有提供计算加权平均值的内置功能，T-VWAP 根据方程通过自行编写的 reduce 函数实现。自行编写的函数可能不如内置函数优化得好。T-VWAP 和 O-S 都使用 InfluxDB 中的两个字段列来计算结果。在 InfluxDB 中，两个字段列分别位于不同的序列中，计算前需要通过透视将它们合并。由于 T-VWAP 和 O-S 在透视操作前都不包含时间窗口聚合操作，因此透视需要处理大量原始数据行。T-VWAP 和 O-S 的查询延迟较高，可能是因为对大量原始数据进行透视操作耗时较长。

6.3.1 结论

- kdb+ 在计算密集型查询方面具有最佳的整体性能。
- 在操作多个字段列且需要进行透视时，InfluxDB 可能会耗费较长时间。

6.4 复杂查询

基准测试 C-R、C-VT、C-VO1 和 C-VO2 用于评估数据库在复杂查询下的性能。表8分别展示了所得结果。总体而言，kdb+ 和 InfluxDB 的查询延迟低于 TimescaleDB 和 ClickHouse。在 C-R 和 C-VO1 中，kdb+ 和 InfluxDB 的结果相近；在 C-VT 中，kdb+ 速度更快，而在 C-VO2 中，InfluxDB 速度更快。此外，本节所有基准测试中，ClickHouse 的表现均优于 TimescaleDB。

在 C-VO2 中，InfluxDB 的速度比 kdb+ 快 2.8 倍；在 C-VO1 中，两者性能相近。C-VO1 和 C-VO2 在数据范围和聚合时间窗口方面有所不同。这进一步印证了第 6.2 节的结论：InfluxDB 在时间窗口聚合函数方面性能更佳。

基准测试 ID	**kdb+ 内存中**	**kdb+ 磁盘上**	**InfluxDB**	**TimescaleDB**	**ClickHouse**
	查询延迟（毫秒）				
C-R	64	113	99	1,614	401
C-VT	41	51	2,009	324	190
C-VO1	61	96	94	1,591	407
C-VO2	-	688	242	4,319	1,139

表8：复杂查询性能

结论

- 复杂查询性能：kdb+ = InfluxDB >ClickHouse >TimescaleDB。
- InfluxDB 在时间窗口聚合函数方面表现出色。

6.5 查询存储

由于数据库内置功能的局限性，仅在 kdb+ 和 InfluxDB 中测量查询执行所占用的存储空间。表 9 展示了通过数据库内置功能获得的查询执行期间存储空间的使用情况。显然，与 kdb+相比，InfluxDB 的内存使用量显著更少。在大多数查询中，kdb+ 使用的内存是 InfluxDB 的 1000 倍。在基准测试 O-S 中，InfluxDB 的内存使用量最多，但仍比 kdb+少 4 倍。前文表明，kdb+ 总体上比 InfluxDB 更快，而这里的结果则显示，InfluxDB 的内存使用效率更高。这表明时间与空间之间存在权衡。将数据和操作移至主存通常能加快查询执行，因为访问主存比访问磁盘快得多。

结论

- 查询所耗费的时间与空间之间存在权衡。

基准测试编号	kdb+ 磁盘上	InfluxDB
	查询存储（MB）	
T-V1	109	0.016
T-V2	83	0.0005
T-VWAP	48	33
O-T	2	0.003
O-B1	0.005	0
O-B2	0.013	0
O-S	588	145
O-V1	184	0.694
O-V2	184	0.153
O-NBBO	0.004	0.035
C-R	109	0.014
C-VT	54	32
C-VO1	109	0.028
C-VO2	109	0.023

表 9：kdb+ 和 InfluxDB 中查询存储的性能

6.6 基准测试评估

根据以上结果和讨论，可以得出结论：对于时间序列数据的金融分析应用，kdb+ 是最合适的数据库。kdb+ 能够快速批量摄取数据，并具有良好的压缩率，从而提升存储效率。对于所有基准测试，包括读取查询、计算密集型查询和复杂查询，kdb+ 的查询延迟都保持稳定且较低。然而，查询时kdb+ 占用的内存空间明显大于InfluxDB。

InfluxDB 的存储效率最佳，但其数据摄取吞吐量明显慢于 kdb+ 和 TimescaleDB。在大多数查询中，InfluxDB 的性能排名第二。不过，InfluxDB 在某些操作中的性能会有所波动。观察发现，在执行时间窗口聚合函数时，InfluxDB 的查询延迟可能低于 kdb+ 。此外，多个字段列上的透视操作可能会显著降低执行速度。

TimescaleDB 的数据摄取表现良好，且压缩率较高。在大多数基准测试中，它的查询延迟最高。

ClickHouse 在数据摄取和存储效率方面表现最差，这可能是因为 ClickHouse 并非针对时间序列数据进行优化。整个基准测试过程中，ClickHouse 的查询性能一直稳定地排名第三。

本项目设计的基准测试套件能够评估和比较数据库系统的性能。评估内容涵盖数据库的不同方面，包括数据摄取、存储效率、读取查询、计算密集型查询和复杂查询。基准测试采用通常用于交易和订单簿数据的计算方法，不依赖于数据库特定的模型或算法。这意味着，只需导入数据并执行按照规格编写的查询，即可轻松将该基准测试套件扩展至其他数据库系统。此外，基准测试结果不会受到数据库特定算法的偏倚。

此外，本项目的结果与相关研究的结果具有可比性。关于 kdb+最适用于金融分析用例的结论，与 [14] 和 [15]的研究结果一致。就数据摄取、存储效率和查询执行而言，InfluxDB 和 TimescaleDB 的测试结果也与 [1]的结果一致。

7 结论

时间序列数据是按照时间顺序记录的属性测量值。时间序列数据不同于传统的关系型数据，因为它通常规模较大、生成频率较高，而且旧数据很少更新。许多数据库系统都针对时间序列数据的这些特点进行了优化。时间序列数据已广泛应用于金融行业，例如交易决策和市场分析等用例。

基准测试可以评估数据库性能，并比较不同数据库的性能。本项目设计了一套适用于金融数据分析用例的基准测试套件，可扩展至多种数据库。这些基准测试已在 kdb+、InfluxDB、TimescaleDB 和 ClickHouse 上实施并测试。收集到的结果从数据库性能的不同方面进行了分析。此外，本研究还评估了所提出的基准测试套件有效评估数据库性能的能力。
首先，定义一个存储和查询时间序列交易数据及订单簿数据的场景，并确定相关用例。随后回顾与这些用例相关的研究。基准测试套件基于文献中总结的常用计算方法和指标进行设计，涵盖数据使用方式、输入/输出强度和计算强度各不相同的工作负载。

在一个月的时间内，通过加密货币交易所的实时数据流收集了总计 25 GB 的数据。批量写入基准测试使用随机数据生成。通过导入数据集并使用各数据库原生的查询语言实现基准测试，在 kdb+、InfluxDB、TimescaleDB 和 ClickHouse 上测试了该基准测试套件。

对实验结果进行了分析，得出的结论是，kdb+ 在四种数据库中总体性能最佳。此外，还从数据摄取与存储、读取查询、计算密集型查询和复杂查询等多个方面对 InfluxDB、TimescaleDB 和 ClickHouse 进行了评估。

该基准测试套件能够展现不同数据库之间的性能差异，所得结论也与先前研究一致。据我们所知，这是首个针对金融分析用例设计的时间序列数据库基准测试。STAC-M3 中的基准测试也面向金融数据库，但不对公众开放。

7.1 局限性

尽管本项目达成了目标，但仍存在一些局限性。基准测试设计的局限在于用例方面。当前设计较为通用，可应用于广泛的用例。若要让基准测试套件更有效地评估数据库性能，就应开发不同的基准测试，以针对数据输入规模和计算复杂度不同的用例。例如，针对使用历史数据执行复杂定价模型的用例所设计的基准测试，应与针对快速实时分析用例的基准测试有所不同。此外，查询执行期间收集的性能指标只有查询延迟。纳入更多指标有助于更全面地评估数据库系统。

受机器的时间和内存空间限制，数据取自一个月的实时数据样本。然而，许多股票市场历史数据分析会使用涵盖多项资产、跨越数月至数年的数据。基于更大规模的数据设计并测试基准测试，可以更真实地模拟计算。硬件方面的另一项局限是，基准测试仅在单线程环境下进行，且只有一个客户端连接到服务器。这意味着该基准测试套件无法在可扩展环境中评估数据库性能。

实验结果可能受到数据库软件的限制。本项目在选定的 4 个数据库上实现了基准测试。此外，本项目使用的数据库软件均为开源社区版本。不过，数据库供应商也提供包含更多功能的付费企业解决方案。这些功能可能会影响数据库性能，并得出不同的结论。

8 未来工作

8.1 设计

为了使设计更贴合用例，可以针对不同用例开发更多基准测试。例如，测试更大规模的实现时，可以包含多个线程，以及同时连接到服务器的众多客户端。应使用由数据库服务器组成的集群和大量客户端机器来评估可扩展系统。

用于构建基准测试的样本计算和分析也可以改进。可以借鉴研究成果中的更复杂分析方法。金融行业的交易员和开发人员可以参与设计过程。他们可以分享工作中常用的算法和数据模型，以及在此用例下对合适数据库系统的要求。

8.2 实验

实验可以扩展至其他常用于类似工作负载的数据库。实验所用的硬件环境也可以扩展至工业级服务器和集群。此外，磁盘、内存和中央处理器等服务器技术也可能影响数据库性能。对更多数据库和服务器进行实验，可以让开发人员全面了解数据库性能。开发人员可以结合实施规模、性能偏好、权衡因素及相关成本，选择合适的数据库。

参考文献

1. Hao Y、Qin X、Chen Y、Li Y、Sun X、Tao Y、Zhang X 和 Du X。TS-Benchmark：时间序列数据库基准测试。*2021*年第*37*届*IEEE*国际数据工程会议（*ICDE*）。2021年：588–99。数字对象唯一标识符：10.1109/ICDE51399.2021.00057 2. STAC。STAC-M3 Central。2022年。可从以下网址获取：https://stacresearch.com/m3 3. Cooper BF、Silberstein A、Tam E、Ramakrishnan R 和 Sears R。使用YCSB对云服务系统进行基准测试。第一届*ACM*云计算研讨会论文集。2010年：143–54 4. Fu Tc。时间序列数据挖掘综述。《人工智能工程应用》2011年；24：164–81 5. Shafer I、Sambasivan RR、Rowe A 和 Ganger GR。大型数值时间序列的专用存储。第五届*USENIX*存储与文件系统热点主题研讨会（*HotStorage13*）。2013年 6. Bitincka L、Ganapathi A、Sorkin S 和 Zhang S。使用半结构化时间序列数据库优化数据分析。通过日志分析与机器学习技术管理系统研讨会（*SLAML 10*）。2010年 7. Przymus P 和 Kaczmarski K。使用图形处理器的时间序列数据库动态压缩策略。数据库与信息系统新趋势。施普林格，2014年：235–44 8. Rhea S、Wang E、Wong E、Atkins E 和 Storer N。LittleTable：时间序列数据库及其应用。*2017*年*ACM*国际数据管理会议。2017年：125–38 9. Pelkonen T、Franklin S、Teller J、Cavallaro P、Huang Q、Meza J 和 Veeraraghavan K。Gorilla：快速、可扩展的内存时间序列数据库。《VLDB基金会会刊》2015年；8：1816–27 10. Adams C、Alonso L、Atkin B、Banning J、Bhola S、Buskens R、Chen M、Chen X、Chung Y、Jia Q 等。Monarch：Google的行星级内存时间序列数据库。《VLDB基金会会刊》2020年；13：3181–94 11. KX。使用kdb+ 和q语言进行开发——Kdb+ 与q文档。2022年。可从以下网址获取：https://code.kx.com/q/ 12. KX。架构——q与kdb+ 文档——Kdb+与q文档。2022年。可从以下网址获取：https://code.kx.com/q/architecture/ 13. Tomczak P。是什么让时间序列数据库kdb+ 如此快速？2019年。可从以下网址获取：https://kx.com/blog/what-makes-time-series-database-kdb-so-fast/ 14. Hyndman H。Kdb+ 传递比较。2018年。可从以下网址获取：https://kx.com/wp-content/uploads/2020/11/KdbTransitive-Comparisons-1.pdf 15. Bodon F。KX的kdb+ 与英特尔傲腾持久内存在STAC-M3金融服务基准测试中创造多项世界纪录。2021年。可从以下网址获取：https://kx.com/blog/kxs-kdb-and-intel-optane-attain-several-world-records-in-stac-m3-

16.Gruen A. 用例：Cobalt 将 Kx 技术与新一代外汇交易后处理网络相结合。2018年。来源：https://kx.com/blog/use-case-cobalt-integrates-kx-technology-with-next-generation-fx-

17.KX. 一家跨国投资银行借助 Google Cloud 上的 KX 实时采集关键市场和交易数据。2022年。来源：https://kx.com/blog/investment-bank-captures-and-processes-1tb-of-data-per-day-with-k

18. KX. KX 助力一家金融科技公司以五分之一的成本实现 10 倍性能。2022年。来源：https://kx.com/blog/fintech-company-achieves-10x-performance-at-1-5th-of-the-cost-wit

19. Plattner H 和 Zeier A.《内存数据管理：技术与应用》。施普林格科学与商业媒体，2012年 20. Plattner H 等。《内存数据管理课程》。施普林格，2013年

金融服务基准测

交易后处理网络

x-on-google-cl

h-kx/

21. Borror JA.《q For Mortals 第三版：q 编程导论》。q4m LLC，2015年 22. Tinnefeld C.《在 RAMCloud 上构建列式数据库：面向低延迟数据中心的数据库设计》。施普林格，2015年 23. Novotny J, Bilokon PA, Galiotos A, 和 D´el`eze F.《使用 kdb+/q 进行机器学习与大数据》。约翰·威立父子公司，2019年 24. InfluxData。《influxdb：开源时间序列数据库》。2022年。来源：https://www.influxdata.com/products/influxdb-overview/ 25. InfluxData。《InfluxDB 设计见解与权衡》。2022年。来源：https://docs.influxdata.com/influxdb/v1.8/concepts/insights_tradeoffs/ 26. Churilo C.《InfluxDB 在时间序列数据与指标基准测试中胜过 Cassandra》。2018年。来源：https://w2.influxdata.com/blog/influxdb-vs-cassandra-time-series/ 27. Churilo C.《在时间序列工作负载方面，InfluxDB 比 MongoDB 快 2.4 倍》。2018年。来源：https://w2.influxdata.com/blog/influxdb-is-27x-faster-vs-mongodb-for-time-series-workloads/ 28. InfluxData。《客户案例：第一资本》。2022年。来源：https://www.influxdata.com/customer/capital-one/ 29. InfluxData。《客户案例：Robinhood》。2022年。来源：https://www.influxdata.com/customer/robinhood/ 30. Naqvi SNZ, Yfantidou S, 和 Zim´anyi E.《时间序列数据库与 InfluxDB》。课程论文，布鲁塞尔自由大学，2017年；12 31. InfluxData。《内存索引与时间结构化合并树（TSM）》。2022年。来源：https://docs.influxdata.com/influxdb/v1.8/concepts/storage_engine/ 32. Inc. T.《简化时间序列数据 — Timescale》。2022年。来源：https://www.timescale.com/ 33. Inc. T.《概览 — Timescale 文档》。2022年。来源：https://docs.timescale.com/timescaledb/latest/overview/#accelerated-performance 34. Inc. T.《分布式超表 — Timescale 文档》。2022年。来源：https://docs.timescale.com/api/latest/distributed-hypertables/ 35. Kiefer R.《TimescaleDB 与 PostgreSQL 的时间序列比较：插入速度提高 20 倍，删除速度提高 2000 倍，查询速度提高 1.2 至 14,000 倍》。2017年。来源：https://www.timescale.com/blog/timescaledb-vs-6a696248104e/ 36. Inc. T.《超表架构 — Timescale 文档》。2022 年。来源：https://docs.timescale.com/timescaledb/latest/overview/core-concepts/hypertables-and-chunks/hypert

37.Inc. C. 快速开源联机分析处理数据库管理系统——ClickHouse。2022年。来源：https://clickhouse.com/ 38.Inc. C. 什么是 ClickHouse？ — ClickHouse 文档。2022年。来源：https://clickhouse.com/docs/en/intro/ 39.Inc. C. ClickHouse 为什么这么快？ — ClickHouse 文档。2022年。来源：https://clickhouse.com/docs/en/faq/general/why-clickhouse-is-so-fast/ 40.Inc. C. ClickBench——分析型数据库管理系统基准测试。2022年。来源：https://benchmark.clickhouse.com/ 41.Inc. C. MergeTree — ClickHouse 文档。2022年。来源：https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/mergetree/#mergetree-data-st

42.Inc. C. ClickHouse 索引设计 — ClickHouse 文档。2022年。来源：https://clickhouse.com/docs/en/guides/improving-query-performance/sparse-primary-indexes/sparse-pr
43. Bonnet P 和 Shasha D. 数据库基准测试。数据库系统百科全书。Liu L 和¨Ozsu MT 编。纽约，纽约：施普林格纽约，2018年：936–8。数字对象唯一标识符：10.1007/978-1-4614-8265-9_80797。可从以下网址获取：https://doi.org/10.1007/978-1-4614-8265-9_8079744. Gray J. 数据库和事务处理系统基准测试手册。摩根·考夫曼出版社，1992年 45. Han R、John LK 和 Zhan J. 大数据系统基准测试：综述。IEEE 服务计算汇刊 2018年；11:580–97。数字对象唯一标识符：10.1109/TSC.2017.2730882 46. Arlitt M、Marwah M、Bellala G、Shah A、Healey J 和 Vandiver B. IoTABench：物联网分析基准测试。第六届 *ACM/SPEC* 国际性能工程会议论文集。2015年：133–44

47. Arasu A、Cherniack M、Galvez E、Maier D、Maskey AS、Ryvkina E、Stonebraker M 和 TibbettsR. Linear Road：流数据管理基准测试。第三十届超大型数据库国际会议论文集——第 *30* 卷。2004年：480–91 48. Murphy B. 什么是 STAC M-3 基准测试，为什么值得关注？2020年。可从以下网址获取：https://www.weka.io/blog/what-is-the-stac-m-3-benchmark-and-why-should-you-care/ 49. Pandey R. 使用 YCSB 对云数据库 MongoDB（NoSQL）与 MySQL（关系型）进行性能基准测试和比较。技术报告。技术报告。https://doi. org/10.13140/RG. 2.2. 10789.32484，2020年 50. Seghier NB 和 Kazar O. 使用 YCSB 工具对 NoSQL 数据库进行性能基准测试和比较：Redis、MongoDB 与 Cassandra。*2021* 年数学与信息学最新进展国际会议（*ICRAMI*）。2021年：1–6。数字对象唯一标识符：10.1109/ICRAMI52622.2021.9585956 51. benchANT. YCSB 基准测试终极指南（2021）。2021年。可从以下网址获取：https://benchant.com/blog/ycsb 52. Patil S、Polte M、Ren K、Tantisiriroj W、Xiao L、L´opez J、Gibson G、Fuchs A 和 Rinaldi B. Ycsb++：对可扩展表存储中的高级功能进行基准测试和性能调试。第二届 *ACM* 云计算研讨会论文集。2011年：1–14 53. Muranaga J、Shimizu T 等。市场微观结构与市场流动性。日本银行，1999年 54. Chandrapala P. 交易量与股票回报之间的关系。《竞争力杂志》2011年 55. Berkowitz SA、Logue DE 和 Noser Jr EA. 纽约证券交易所的交易总成本。《金融学杂志》1988年；43:97–112 56. Cao C、Hansch O 和 Wang X. 纯限价订单簿市场中的订单下达策略。《金融研究杂志》2008年；31:113–40 57. Abhyankar A、Ghosh D、Levin E 和 Limmack R. 买卖价差、交易量与波动率：伦敦证券交易所的日内证据。《商业金融与会计杂志》1997年；24:343–62 58. Cao C、Hansch O 和 Wang X. 公开限价订单簿的信息含量。《期货市场杂志：期货、期权及其他衍生产品》2009年；29:16–41 59. Moscon B. Cryptofeed。2022年。可从以下网址获取：https://github.com/bmoscon/cryptofeed 60. DEVnet. qPython。2022年。可从以下网址获取：https://github.com/exxeleron/qPython