華中科技大學

论文中期进展报告

题 目：基于特征编码的电力时序数据存储与多维索引研究

学　　号　M202273662

姓　　名　金灿果

专　　业　计算机科学与技术

指 导 教 师　左琼

院（系、所）　计算机科学与技术学院

华中科技大学研究生院制

填 表 注 意 事 项

一、本表适用于攻读硕士学位研究生选题报告、学术报告，攻读专业硕士学位研究生实践环节报告，攻读博士学位研究生文献综述、选题报告、论文中期进展报告、学术报告等。

二、以上各报告内容及要求由相关院（系、所）做具体要求。

三、以上各报告均须存入研究生个人学籍档案。

四、本表填写要求文句通顺、内容明确、字迹工整。

1. 课题来源

1.1 研究背景

随着大数据和物联网技术的快速发展，全球数据量急剧增长，特别是在金融、环境检测、医疗生物、工业制造、农业生产、软硬件系统平台等领域，时间序列数据的产生和应用日益广泛。这些领域对时序数据的处理提出了更高的扩展性要求，高效管理时序数据对各领域至关重要。电力系统作为与社会民生紧密相关的基础设施，大规模电力设备时时刻刻产生着各种状态感知信息，包括分布在数据采集和监控系统（Supervisory Control and Data Acquisition，SCADA）、广域量测系统（Wide Area Measurement System，WAMS）、地理信息系统（Geographic Information System，GIS）和故障录波系统（Fault Information System，FIS）中的各类静态和动态数据[1]，这些多源异构的电网时序数据为电网安全决策分析与故障智能诊断提供了丰富的数据源。

电网的智能化建设需要全方位对电网设备运行状态、用电信息、故障信号等进行实时检测、分析，数据采集和采集频率会大幅度增加。目前电力系统普遍做法是基于 HBase、MongoDB 等大数据平台的研究和应用[2][3]，使用时序数据库软件作为实时数据存储和读取的工具[2]。

随着大模型与数据库的发展结合，数据库上的查询需求也越来越丰富，索引机制从传统的维度单一化向多维特征感知、智能化方向发展。特别是在电力数据管理领域，需要设计更加专业的索引结构，以满足不同的应用场景需求。

1.2 研究意义

基于 HBase、MongoDB 等大数据平台的解决方案在智能电网数据管理中，需要对电力调度数据进行计算，这类数据具有数据量大、数据类型复杂、价值密度较低、处理时限较短[3]等特点。电力故障时，需要利用数据采集系统得到故障时刻、故障设备、故障类型、故障演进过程等进行分析。单一数据源进行诊断会受到各自数据源本身缺点的影响，将多种数据源信息进行整合再进行诊断，实现各数据源信息的互补，成为故障诊断新方向。

海量电力数据系统分析中，正常数据在数量规模上远超异常数据，数据压缩后访问数据时，需要解压缩大量无关数据，导致异常数据分析时遍历到的无关数

据量多，影响访问速度，实时处理能力受限[4]。

WAMS 和 SCADA 数据作为数据源的故障针对时不可避免受制与不统一的时间标尺，因而时间信息在索引构建时需要保留。传统关系型数据库例如通常以键值的形式构造索引，针对复杂查询例如多维查询、范围查询、K 近邻查询，这样的索引方案难以满足 TB 数量级数据访问的时效性[5]。

电力数据系统采用批量更新机制，多个维度的数据会同步发生变化，并以流式数据传输方式确保数据的高效处理与实时交互，要实现电力系统海量时序数据的高效管理与分析，核心需求体现在数据特性方面与性能指标方面。数据特性上需要满足高并发读写、实时处理、多维关联与高效压缩；性能指标上需要有快速检索响应能力，实时更新能力，存储空间上的优化以及查询效率的保障。

2. 课题研究内容和技术方案

2.1 课题研究内容

面对大规模、高频率的电力系统多维时序数据，现阶段的关系型数据库在写入时可能会遇到锁竞争、事务延迟等问题，导致写入性能下降，时序数据库针对高频率写入进行优化，时序数据形态各异，例如突刺变化、折点变化、周期变化等诸多形态，阈值范围也各有不同[6]，对于复杂形态下的异常，很难在电力数据的存储基础上进行灵活的数据故障预警、故障事件的查询与连锁故障分析。本课题旨在从多维电力时序数据特征出发，设计面向电力时序多维数据特征的存储与索引研究的组织方案，提升数据管理效率，在查询性能提高的同时减少数据存储空间，并支持实时复杂查询。具体包括以下几个方面的研究内容：

（1）电力数据特征分类和基于特征编码的数据模型

研究 WAMS 和 SCADA 系统中不同组件和传感器所产生的多源数据，针对数据特征对正常和多种异常数据进行分类。针对智能电网多维时序数据存储与检索的需求，提出基于特征识别的数据编码模型。

（2）特征编码的多维索引树

针对多维时序数据管理上会出现的复杂查询需求，为了快速定位故障时时刻和位置，提出一种基于特征编码的多维索引树结构，其中包括索引树结构的设计、索引树构建算法、索引维护机制。索引树在传统的 B+树上进行了拓展，满足聚合查询、范围查询，并且突破时序数据库索引仅基于维度的局限性，在设计时考

虑到时序数值变化的相似性，以支持任意长度的相似性查询。

（3）基于精度的浮点数擦除方案改进

针对数据利用价值和使用需求，在目前的流式浮点数压缩算法上进行改良，从精度上调整压缩程度，需要保留的精度越高，压缩比越小。改进算法较原算法更符合块级压缩存储，并且在时间开销上有一定的提升。

（4）基于特征编码的压缩策略

考虑到电力不同类型的数据变化特点不同，为降低电力数据存储成本的同时尽可能保留对电力系统运行状态分析的关键信息。提出基于块特征自动选择压缩方式的块压缩策略，在存储时保留块的特征。定义候选算法的评价指标，按照候选算法的计算得分选择压缩算法。并且依赖不同特征数据的使用频率不同定义不同特征的评价指标权重，以最大程度利用数据特征。

（5）性能评估与验证

验证效果的有效性，存储上与现有的传统数据库、时序数据库上压缩算法进行压缩效率、空间利用率、写入性能的对比，查询上对比响应时间、查询吞吐量、查询准确率。压缩编码方面与现有的无损压缩例如流式数据压缩、基于字典的编码压缩对比压缩时间、存储空间、CPU 和内存的开销。分析实验对比结果，以评价数据模型、索引设计以及压缩算法的性能情况和改进程度。

2.2 技术方案

2.2.1 电力数据存储设计

2.2.1.1 现有的电力数据存储设计

分布式数据库 HBase 是 Hadoop 成为主流大数据框架体系背景下的基于列存储的数据库，由于其稀疏、持久化、分布式、面向列族的键值对读写、多维映射等特性，以及在调度数据时的实时性和高速读写的特点，HBase 成为在电力数据存储研究中常用到的数据库。HBase 中的数据存储在表中，物理上被分割为多个 Region。行键是唯一标识，按字典排序。物理上按列族存储，创建时列族自定义，在插入数据时动态添加。HBase 在键值存储中会存在数据源的冗余和采集指标的冗余，单一的索引和表结构导致复杂查询问题时需要联合多个查询进行分析，影响查询性能[7]。

常用的关系型数据库管理在时序数据压缩时会存在数据量大维护成本高、即

时处理分析数据性能差的问题，于是产生了基于时序数据特征设计的数据库管理系统。以 OpenTSDB、KairosDB[8]为代表的基于分布式存储的时间序列数据管理将每个条目都与时间戳相关联，保持一段时间内系列值的连续性，以时序数据为单位，根据时序特征对数据进行压缩。

但是这类数据库会存在一些不足，例如低效的唯一标识符机制，依赖 Hadoop 和 HBase 环境、部署及维护成本高。以 InfluxDB[9]为代表的垂直型时序数据库成为时序数据库市场的主流，采用类似日志结构合并树（Log-Structured Merge Tree，LSM Tree）的时序合并树（Time Series Merge Tree，TSM Tree）存储结构，引入时序键的概念，同一个时序键中的时间线数据写入同一个数据块中，而同一个数据块内的数据则属于同一个数据源下的同一字段，减少了冗余存储。InfluxDB 的数据采用 LSM Tree 结构将数据写入内存，写入内存之后按时序键组织。在复杂查询例如相似性查询上需要对多维数据进行大范围遍历，基于时间上的索引作用收到了限制，获取块数据时需要对整块数据进行解压缩，获取无关数据较多。

Prometheus[10]使用标签（Labels）和度量（Metrics）来组织时序数据，标签时键值对的形式被索引，通过标签添加多个维度。TimescalaDB[11]通过分区和超表的概念支持按照不同维度对数据进行分区，允许用户在多个维度对数据进行水平切分，提供连续聚合的功能，预先计算和存储常用的聚合结果。但处理高维度数据时，分区策略不够灵活。OpenTSDB 提供多个内置聚合器，在查询中使用聚合器进行多维聚合操作。这些时序数据库都是基于时间范围进行的索引，维度是建立单独的索引结构，在面对复杂查询时，对高维度的索引有限。

2.2.1.2 查询需求分析

SCADA 系统的数据主要包括温度、压力、流量、电流、电压等传感器测量值，WAMS 系统的数据主要包括电压幅值、相位角、频率等相量数据以及功率、电流、无功功率、有功功率等数据。将 WAMS 和 SCADA 系统上的查询需求划分，可以分为以下三种查询：

聚合查询：在运行监控中进行电压统计分析，电流趋势分析，电能质量评估等实时性分析需求。在设备管理中，需要通过聚合查询来进行负载分布统计，效率指标汇总与性能评估分析。同时多维度聚合计算，不同时间粒度的统计和复杂的电力指标组合分析也给电力系统上的聚合查询带来挑战。

范围查询：在异常识别过程中需要进行故障定位，获取故障前状态、故障过程、故障后影响等等。也可通过查询的进行历史对比，周期评估和数据趋势跟踪来进行运行分析。由于是监测的海量大规模时序数据，因而需要高精度的时间定位，同时要保证快速响应要求。

相似性查询：模式识别场景在带有特征的电力数据上非常常见，例如故障模式识别中的典型故障模式、异常波动特征、连锁故障模式。另外在连锁故障分析时，也可以通过数据变化的相似性判断异常可能存在的关联性。相似性查询除了模式匹配外还有序列相似度、特征关联性以及时间弹性。涉及到多维度时要考虑多维数据关联性，同样也要保证匹配精度，实时响应和可扩展性。

2.2.1.3 系统总体架构设计

本文提出一种基于特征编码的 WAMS 数据管理系统，采用分层架构设计，包括数据接入层、数据编码层、存储管理层和查询服务层四个核心层次。本节介绍各层次的功能设计和关键技术。

流式数据经过特征分类和采集获取特征编码，作为信息摘要存储在数据块的头部并作用于数据块压缩算法的选择。采用块级存储方式块的存储头部包含时间信息和索引信息，数据部分只存储采集数据值，这样可能尽可能消除时间戳信息、维度信息的冗余。基于特征的存储替代键值对存储，块级别也利用特征压缩。利用特征编码替代标签，采用自适应的压缩策略，空间占用优化从而提高了存储效率。

实现特征感知结合数据范围的快速过滤。将维度信息编码设计于索引的存储中，满足时序数据批量更新的时间属性要求，增加节点操作简单，支持多维度并行检索，能在压缩数据上直接查询，减少了不必要的数据存储。系统架构如图 2 所示。

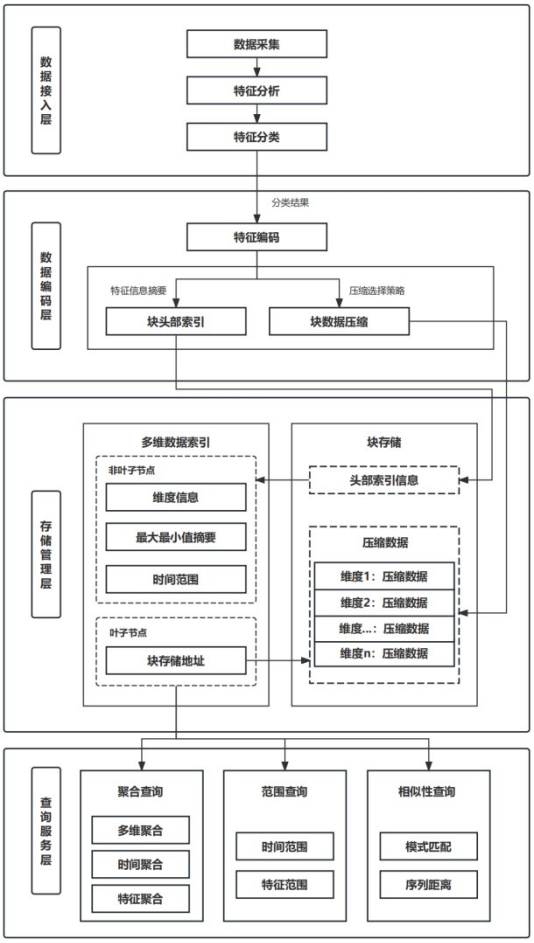

图 1.数据架构图

（1）数据接入层

负责 WAMS 数据的采集，特征分析，特征识别分类。特征分析部分将对电力数据异常分类，并基于不同类别的异常定义特征提取方法。数据采集后，按照时间窗口对数据划分，每个窗口内的数据进行特征分类，分类结果传入数据编码层供后续特征处理。

（2）数据编码层

根据数据接入层的的电力数据特征分类结果，设计特征编码机制，提出浮点数尾数擦除压缩算法改进和基于特征的压缩算法选择策略。每个窗口内的特征编码信息作为窗口信息摘要，同时以每个维度下的划分窗口为压缩单位，根据维度特征编码对应的压缩比、压缩时间、CPU 性能、内存性能的评价指标，结合不同压缩算法的性能比较，进行压缩算法的选择。

（3）存储管理层

存储管理层包括块存储和多维数据索引。数据编码层的编码信息组成块存储部分的块头部存储索引信息，索引信息中有维度信息、最大最小值信息的摘要。块内每个维度的每个时间窗口按照特征划分情况选择压缩算法压缩存储。多维索引树建立在 B+树的基础上，非叶子节点存储块的头部信息叶子节点指向数据块存储地址，用索引过滤符合查询条件的结果集，返回到块存储的地址进行解压缩获取原始数据，以支持查询服务。

（4）查询服务层

查询服务层主要包括聚合查询、范围查询和相似性查询。聚合查询包括多维聚合、时间聚合、特征聚合。时间范围查询包括时间范围查询、数值范围查询、特征范围查询。相似性查询包括模式匹配、序列相似度。

2.2.1.4 电力数据特征分类

基于电力系统不同运行状态的特征表现，将电力数据分为以下七类：

（1）正常（Normal Operation）

在正常运行状态下，电力数据呈现出特定的变化趋势与特征变化。从变化趋势来看，各项指标维持在正常区间内，波动微小且呈现稳定态势。就特征变化而言，其最大值与最小值处于正常范围，均值稳定于预期值，方差和标准差较小，这体现出数据波动程度低。此外，峰度和偏度接近于零，表明数据分布趋近于正态分布。

（2）三相短路（Three - Phase Short Circuit）

三相短路状态下，数据的变化趋势表现为短路相之间的电压急剧降低至零或极低水平，同时电流急剧上升。在特征变化方面，电压的最小值显著下降而电流最大值显著上升，电压均值下降且电流均值上升。方差和标准差显著增大，意味着波动剧烈。峰度增加，表明波形尖峰增多，偏度则依据波形变化方向，可能增加或减少。

（3）相间短路（Phase - to - Phase Short Circuit）

当发生相间短路时，变化趋势体现为两相之间短路，受影响相的电压下降而电流增加。其特征变化为受影响相的电压最小值显著降低，电流最大值显著升高，电压均值下降且电流均值上升。方差和标准差显著增加，反映出波动的剧烈程度。峰度增加表明波形尖峰增加，偏度取决于波形变化方向，可能增加或减少。

（4）相间接地（Phase - to - Ground Short Circuit）

在相间接地情况中，变化趋势是一相与地之间短路，受影响相电压下降且电流增加。特征变化上，受影响相电压最小值显著下降，电流最大值显著上升，电压均值下降且电流均值上升。方差和标准差显著增加，显示出强烈的波动性。峰度增加表明波形尖峰增加，偏度取决于波形变化方向，可能增加或减少。

（5）单相接地（Single - Phase Ground Fault）

单相接地时，变化趋势为一相电压下降，同时流经地面的电流增加。其特征变化是受影响相电压最小值显著下降，地面电流最大值显著上升，电压均值下降且地面电流均值上升。方差和标准差显著增加，表明波动剧烈。峰度增加意味着波形尖峰增加，偏度取决于波形变化方向，可能增加或减少。

（6）断路器断开（Circuit Breaker Open）

断路器断开状态下，变化趋势表现为电路中电流中断，电压可能出现瞬时变化。从特征变化角度看，电流最小值降为零，电压可能有瞬时最大值或最小值。电流均值下降，电压均值可能有瞬时变化。方差和标准差在短时间内增加，随后电流的方差和标准差变为零。峰度瞬时增加后恢复正常，偏度在短时间内可能增加然后恢复正常。

（7）次同步振荡（Sub - Synchronous Oscillation）

次同步振荡状态下，出现频率低于工频（通常在 10Hz 到 2kHz 之间）的振荡现象，导致电压和电流波动。其变化趋势表现为在振荡周期内电压和电流上下波动。在特征变化方面，最大值和最小值在振荡周期内波动，均值在振荡过程中可能变化不大。方差和标准差增加，表明波动性增强。峰度增加表明振荡致使尖峰增加，偏度取决于振荡方向，可能增加或减少。

可形式化表示为：设厂站 A 包含 n 个维度：$(C_1, C_2, C_3, ..., C_n)$在时间周期 T 内，维度 i 的数据序列表示为：$X\_i = \{x_1, x_2, x_3, ..., x_t\}$，其中 Xt 表示在时刻 t 的浮点数值。基于 WAMS 数据的时序特征，建立特征分类体系，如下表 1 所示：

表 1.数据特征分类

特征类别	数据表现形式	检测方法	参数范围	存储结构

平稳	数值在基准值附近波动：$x(t) \in [base \pm \delta]$	滑动窗口方差检测	基准偏差：±0.5% 方差阈值：0.01 窗口大小：1200ms	特征头部 最小值 最大值 基准值 方差
斜坡	数据逐渐增加或减少：$x(t) = x(0) + kt$	斜率检测	变化率：<0% 斜率范围：±0.1%/s 持续时间：>1s	特征头部 最小值 最大值 斜率 持续时间
突变	瞬时值剧变：$\|dx/dt\| > threshold$	导数阈值检测	上升时间：<10ms 变化率：>80% 持续时间：<100ms	特征头部 最小值 最大值 时间戳 峰值 持续时间
断开	数值快速衰减至零	阈值检测	最终值：<0.1% 衰减率：>95% 检测窗口：50ms	特征头部 最小值 最大值 时间戳 衰减率 状态标记
周期波动	正弦波特征：{1.0→0.0→-1.0→0.0}	FFT 频谱分析	幅值：[-1.0, 1.0] 频率：45-55Hz 相位：0-2 π	特征头部 最小值 最大值 频率值 幅值 相位

低频振荡	工频以下振荡：f < 50Hz	小波变换分析	频率：0.1-30Hz 能量：>0.2 持续性：>1s	特征头部 最小值 最大值 频率段 能量值 时间序列

正常运行对应平稳、斜坡或周期波动状态；三相短路对应电流急剧上升、电压急剧降低，并且变化率最大；相间短路对应两相电压斜坡降低，相应电流急剧上升，变化率次之；相间接地对应单向电压斜坡降低，接地电流上升，变化率适中；单相接地对应单相电压斜坡降低，接地电流增加，变化率较小；断路器断开对应电流断开态，电压突变态，后续电流保持零值；次同步振荡与周期波动区分开，低于工频。

维度编码采用类别加关键数据信息的结构编码压缩存储。特征类别使用 1 字节存储特征类型，预留空间可扩展新特征类型。数据结构统一用 8 字节的头部结构，使用 double（8 字节）存储关键数值，float（4 字节）存储次要参数，1 字节存储状态标记。

2.2.2 基于特征存储的电力数据模型

针对 WAMS 和 SCADA 多维时序数据的特征，本节提出一种高效的数据块存储结构，通过特征编码压缩和优化的物理布局实现高效的数据访问与查询。

给定 WAMS 时序数据流 D，包含 n 个维度{C_1, C_2, ..., C_n} ，在时间区间 T 内的数据需进行编码存储。时间区间 T 被等分为 W 个时间窗口：

W_i = [t_{start}^i, t_{end}^i], i ∈ [1,W]。其中：W_i 表示第 i 个时间窗口 t_{start}^i, t_{end}^i 分别表示窗口的起止时间，窗口大小 Delta_t = t_{end}^i - t_{start}^i 保持固定。对于维度 C_i ，其数据表示为三元组：C_i = {Min_i, Max_i, E_i}。其中：Min_i, Max_i 表示该维度在时间区间 T 内的值域范围，E_i = {e_1, e_2, ..., e_W}表示各时间窗口的特征编码序列 e_j = {000,001,010,011,100} 表示特征编码集合。

2.2.2.1 数据库上的存储结构

在数据库和存储系统中，索引方式的选择对性能和存储效率有重要影响。

Bigtable 和 Apache Cassandra 都是基于 SSTable（Sorted String Table）有序字符串表进行存储的，数据按键排序，由多个段组成，段由索引文件，存储键和对应的偏移量，通过二分查找在索引文件中定位目标键，读取对应数据。写入数据时需要排序和合并，可能导致较高的写入延迟，并且索引文件加载到内存中，对于大规模数据来说内存占用较高。Hash 索引也是存储中常见的索引技术，利用哈希函数将键值映射到存储位置，但不同的键可能映射到同一个槽，并且哈希索引不支持有序的区间查询，只适用于点查询。

块索引方式是基于块的索引技术，通常用于文件系统和数据库中，数据被划分为固定大小的块，块包含多个记录和数据项，块索引存储每个块的起始地址和元数据，方便快速定位目标块，适用于大规模数据存储。但是块内搜索通常时线性的，对大块内的数据查找效率较低。并且块索引需要定期维护，特别是在数据频繁更新的时候。

2.2.2.2 数据块结构设计

电力数据采用批量式的更新方法，因而插入删除操作相对较少，更多的是增量型操作。电力数据日汇聚量高达 10GB 导致存储规模较大，但是块存储时，块内数据查找困难，因而在设计时采用块内分层，将块内存储的维度和维度信息按索引的形式存储在头部文件中，在块过滤的同时能对块内信息进一步过滤。

数据块采用分层、按时间范围进行组织，如图 2 所示，主要包含头部信息和维度数据、编码存储区域三个部分。块头部结构包含起始时间戳，结束时间戳，特征位图，维度数量，窗口数量，全局最小值和全局最大值。维度表结构包括维度 ID，维度最小值，维度最大值，编码偏移量，编码长度。

图 2.数据块结构

块头部固定 64 字节，满足缓存行对齐，维度表项按照 8 字节边界对齐，编码区域按页面大小对齐。维度采用编码压缩，包括基础编码值以及压缩编码序列，压缩编码序列。维度表按顺序存储，支持二分查找；编码区域连续存储，支持批量读取；关键字段缓存，减少 I/O 操作。

2.2.3 基于特征编码的多维时序数据索引设计

根据 2.2.2.1 提到的电力系统聚合查询、范围查询和相似性查询的需求，提出多维时序数据的索引设计。

2.2.3.1 现有的多维数据索引方法

R 树[12]、KD 树[13]、Quadtree[14]这类的多维空间树型索引结构，按照维度划分空间进行存储，但是维度大量增加，性能就会显著降低，维度扩展时性能也会下降。面向电力数据与时间强关联型的存储时，节点插入还要考虑维度的比较，并且插入删除操作树的重平衡操作会过多，动态更新困难，不适用于批流式的电力数据插入。

Grid-file[15]是将多维空间划分到多个网格单元，通过网格索引快速定位包含目标数据的网格单元。电力时序数据主要是基于时间维度连续写入，Grid-file 这样多维均匀分割特性与时序数据的写入模式不匹配。并且随着新数据的持续写入，可能会导致网格频繁划分。当时序数据维度密集写入时，会导致严重的数据倾斜，或是某些网格单元过载而其他空置的情况。

2.2.3.2 现有的相似性索引方法

相似数据流的搜索方法都是基于相似性算法对数据进行比较，通常是等长比较或是降维再对比[16]。

降维是将数据序列划分成多段，每段用一个概要信息描述，在相似时间序列匹配时已有的方法多采用基于距离的索引[17]。这种方法无法满足按照时间分区和高效的时间窗口查询，划分方法通常是基于等长段的划分或是提取固定长度的段来代表子序列的特征。在相似时间序列匹配时基于距离的索引需要序列距离的参照，面对流式数据的时刻更新，索引会出现频繁插入和更新的情况。基于降维的方法会导致信息损失从而使得多元时间序列中的时序特征缺失造成序列相似性匹配不准确。而如果考虑数据流中所有维度的相似性特征匹配，不仅会增加匹配的开销，当电力数据波动延迟时还会影响判断结果[18]。

电力数据上数据特征内存在序列的相似性，依赖特征去建立相似性索引可以从特征维度过滤掉无关数据。同时特征上的摘要信息在供聚合分析的同时也可作为距离判断信息对维度进行剪枝。流式数据来临时可通过索引节点的插入判断实时进行异常分析。

现有的时序数据根据设置 pattern 的匹配项作为查询，以及额外的平均值范围作为约束条件，以搜素时序值在一定范围的子序列，例如 TS-index[21]。但是这种筛选范围和阈值都很大，面对电力数据低频振荡、周期波动这类值上相似的数据无法察觉，会导致结果集较多或者出现误报结果。现有子序列相似性索引，大部分是定长的查询，例如 isax[22]，tardis[23]等索引，针对不定长的查询时，索引的区间设计往往未考虑数据变化的特点，例如 kv-index[24]索引，是目前最新研究里子序列相似性查询可以针对不定长查询条件的索引方案。这类方法没有考虑到监测数据中对于相邻的子序列，形状差距可能非常小。监测设备时序点过多，子序列的时间重复度高。并且没有考虑多维数据情况下应该如何组织[25]，面对复杂情况的多维属性查询问题，性能会受到显著影响。

2.2.3.3 相似性查询分类

给定时间序列集合 TS = {ts1, ts2, ..., tsn}，查询序列 Q，相似性阈值 ε，定义相似性查询类型如下：

（一）基于查询序列长度的分类

分为全序列匹配和子序列匹配两种方式[17]。全序列匹配即查询序列和被查询序列长度相同，子序列匹配即在较长的时间序列中找到与查询序列相似性的子序列，$QueryType = \{ FullSequenceMatch(Q, TS) = \{ts \in TS \mid len(Q) = len(ts), Similarity(Q, ts) \geq \varepsilon\}, SubSequenceMatch(Q, TS) = \{ sub\_ts \subseteq ts, ts \in TS \mid len(Q) = len(sub\_ts), Similarity(Q, sub\_ts) \geq \varepsilon\}\}$。

（二）基于查询匹配程度的分类

可以分为近似相似性查询和精确相似性查询。近似相似性查询不要求完全匹配，而是寻找在一定误差范围内满足相似性度量阈值的结果，例如 isax、tardis。精确查询则要求结果与查询条件完全匹配。$MatchType = \{ ExactMatch(Q, TS) = \{ts \in TS \mid Distance(Q, ts) = 0\}, ApproximateMatch(Q, TS, \varepsilon) = \{ ts \in TS \mid Distance(Q, ts) \leqslant \varepsilon \}\}$。

（三）相似性度量方式

分为等长的支持相似性度量和支持不等长的相似性度量。等长的相似性度量例如切比雪夫、欧氏距离、曼哈顿距离、余弦相似度：不等长的相似性度量方式有动态时间规整 DTW[19]、最长公共子序列 LCSS[20]、编辑距离。

2.2.3.4 特征索引树设计

索引树采用改进后的 B+树进行组织，在保持 B+树高效范围查询特性的基础上，扩展了节点的数据结构以支持多维特征索引，基于时间属性的存储方式保留时间上的顺序关系。索引的节点结构如下：

索引的头部结构，包括根节点的位置，节点总数，叶子节点数，创建时间戳和更新时间戳。

内部节点结构，包括节点类型，树层级，子节点数量，节点标识，时间范围，特征位图，窗口编码，子节点指针。

叶子节点，包括节点类型，维度数量，节点标识，时间范围，维度数据，下一叶子节点。

索引树的构建采用自底向上的方式进行，结构如图 3 所示。首先将时间序列数据划分基本时间窗口，计算每个窗口的特征编码，将连续的窗口组合形成叶子节点。自上而下构建可以有效减少树的构建开销，在大规模高维数据上的索引节点插入简单。

每个叶子节点存储特定时间范围内的数据，包含范围内的时间窗口编码序列，记录各维度的最大最小值。解决块存储中序列遍历复杂的问题，在叶子节点直接进行维度的定位，避免维度索引的遍历。

基于叶子节点逐层构建内部节点，每个内部节点维护子节点的时间范围聚合信息，特征位图，维度数据的统计信息。保留了时间序列索引的特性，降低了数据处理复杂度，在维度属性查询上也具备索引剪枝能力。

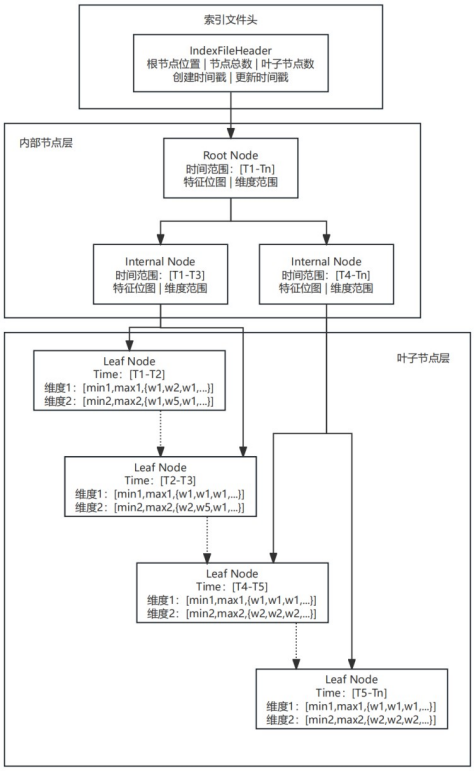

图 3. **多维索引结构**

2.2.3.5 索引维护机制

（1）更新操作。索引树支持两种更新模式：单条数据更新和批量更新。单条数据更新时，通过时间范围定位目标叶子节点，从根节点开始，根据时间范围查找目标叶子节点，确定数据影响的时间窗口。执行更新，更新叶子节点中的特征编码，调整维度的最大最小值，若特征发生变化，更新特征位图。如果节点溢出，执行分裂操作，向上更新父节点的信息，调整树平衡。批量更新时，首先将批量更新数据按时间范围分组，合并可以一起处理的更新操作。对不同时间范围的更新并行处理，使用锁机制确保并发安全，批量更新节点信息。

（2）删除操作。索引树的删除操作支持单条数据删除和批量删除。单挑数据删除时，首先定位删除位置，查找包含目标时间范围的叶子节点，确定需要删除的时间窗口。执行删除时，移除对应的特征，更新节点的时间范围和特征信息，计算维度最大最小值。最后还需要处理节点下溢情况，合并节点以及更新父节点信息。批量删除时对删除的时间范围进行排序，合并重叠的删除范围。找到所有受影响的叶子节点，批量删除数据，重新组织树结构。

所有的操作都通过严格的并发控制机制确保数据一致性，同时采用适当的缓

存策略提高访问效率。

2.2.3.6 基于特征索引段的查询

（1）异常特征查询

针对电力系统中的异常特征检测需求，提出一种基于特征编码的多级过滤查询机制。该机制充分利用索引树的层次结构，实现异常特征的快速定位和精确检索。

给定查询条件 Q = {T, F, D, R}，其中 T = [ts, te]表示时间范围，F = {f1, f2, ..., fm}表示目标特征码集合，D = {d1, d2, ..., dn}表示目标维度集合，R = [vmin, vmax]表示数值范围。异常特征查询过程可形式化描述如下：

异常特征查询采用自顶向下的多级过滤策略，主要包含以下三个层次的过滤机制：

根节点过滤。设根节点 root 的时间范围为 Troot，特征位图为 Broot，则初始过滤条件为：$T \cap Troot \neq \square$，$F \cap getBits(Broot) \neq \square$，$\exists d \in D$, checkRange(d, R) = true。首先检查查询的时间范围是否与根节点的时间范围存在交集，通过特征位图快速判断目标特征是否可能存在，然后利用维度统计信息进行初步的数值范围过滤。

内部节点过滤。对于每个内部节点 N，其过滤条件定义为：TimeFilter(N) = $T \cap N.time\_range \neq \varnothing$，FeatureFilter(N) = $F \cap getBits(N.feature\_bitmap) \neq \varnothing$，DimensionFilter(N) = $\exists d \in D$, $d.range \cap R \neq \varnothing$。根据时间范围逐层向下遍历内部节点，利用节点的特征位图进行特征码匹配，基于节点维度的统计信息（最大值、最小值）进行范围过滤，剪除不满足查询条件的子树分支。

叶子节点精确匹配。对于满足过滤条件的叶子节点 L，提取特征序列 S = {s1, s2, ..., sk}，其中 si 表示时间窗口 i 的特征编码。匹配条件为：$\exists i, si \in F$，$L.dimension\_value \in R$，$L.dimension\_id \in D$。

对满足条件的叶子节点进行详细检查，提取符合特征编码的时间窗口数据，验证数值是否满足查询条件，收集完整的异常特征信息。

（2）不定长子序列相似性查询

给定查询序列 Q 和时间序列数据库 D，不定长相似性查询问题定义：给定查询序列 $Q = (q_1, q_2, ..., q_n)$，目标序列集合 $D = \{S \mid S = (s_1, s_2, ..., s_m), m$

$\geqslant$ n}，qi 和 si 为 Q 和 S 都被划分为窗口大小为 w 的子序列的表示。相似性阈值 ε，要满足查询条件 Distance(S, Q) $\leqslant$ ε，即需要满足约束 μS$\in[\mu Q-\frac{\varepsilon}{\sqrt{w}},\mu Q+\frac{\varepsilon}{\sqrt{w}}]$，σS$\in[\sigma Q-\frac{\varepsilon}{\sqrt{w}},\sigma Q+\frac{\varepsilon}{\sqrt{w}}]$。其中 μS 为候选序列均值，μQ 为查询序列均值[26]。

基于均值约束条件 μS$\in[\mu Q-\frac{\varepsilon}{\sqrt{w}},\mu Q+\frac{\varepsilon}{\sqrt{w}}]$ 进行过滤。多级索引过滤机制包括均值区间过滤 MeanFilter，标准差过滤 StdFilter，数值范围过滤 RangeFilter，动态时间规整过滤 DTWFilter。

不定长匹配时，首先根据均值期间进行过滤，然后通过多级过滤获得更精确的候选集。最后从候选集中通过距离计算公式返回符合相似性条件的结果集。索引时间复杂度分析如表 2 所示：

表 2.**索引时间复杂度**

操作	时间复杂度	说明
特征查询	0(log n)	基于特征编码的树形检索
时间范围查询	0(log n)	基于时间范围的二分查找
维度过滤	0(1)	位图运算实现快速过滤
组合查询	0(log n)	多条件联合查询

2.2.4 数据编码压缩

在对 WAMS 时序数据的处理过程中，由于其采用等间隔采样，且数据类型为浮点数，这为数据的编码压缩提供了特定的条件和方向。鉴于时间戳间隔固定这一特性，在数据块存储方面仅需保存起始时间戳和节点数量即可，在最大程度上减少了数据存储的冗余信息。在此基础上，本节着重提出基于浮点数的编码与压缩方案，旨在进一步提高数据处理效率和存储效率。

2.2.4.1 现有浮点数编码方案

基于数据的统计特性和重复模式的利用，压缩算法分为无损和有损两大类，无损压缩算法通过消除数据中的冗余和重复数据来压缩数据，同时保持数据的完整性和准确性，例如哈夫曼编码、字典编码、预测编码（算数编码、差分编码）等。有损编码则是舍弃数据中的冗余信息来实现更高的压缩率，通常应用在图像、音频和视频等多媒体的数据压缩，例如转换编码、量化、基于模型的压缩等。

目前很多数据库已经用到了这些从压缩解压缩速度上看更为先进且具备通用性的流式浮点数压缩算法，例如键值数据库中 HBase 数据库支持 GZip、Snappy、

LZO、LZ4 算法，时序数据库中例如 InfluxDB 支持 Gorilla[27]、Snappy 的压缩。

针对这类对数据存取速率有较高要求的时序数据，目前最新的做法第一个提出的是 Gorilla 算法，针对 8 字节的 float 进行压缩，将二进制浮点数利用差分编码 XOR 的方式，计算前后相邻数据的二进制异或值，压缩过程就是一个字节流按照一定格式不断追加。Gorilla 使用标记“0”、“10”和“11”来表示三种不同的情况。首数据存储完整的二进制，从第二个数据开始，与前一数据进行异或操作。如果两数相等，异或为 0，只用存储一个比特‘0’。如果异或值不为 0，那么首先会存储控制位‘1’代表异或值不为 0，然后计算前导零个数，有效位个数，有效位比特。如果当前数的有效位数小于前一个数的有效位数，那么可以复用上一个数的有效位数。如果有意义的位块落在前一个有意义的位块内，也就是说，前导零和末尾零的个数至少与前一个值相同，则使用该信息作为块位置，存储控制位‘0’和有意义的 XORed 值。否则，存储控制位‘1’以及后面 5 位的前导零的长度，然后在后面 6 位存储有意义的 XORed 值的长度，最后存储 XORed 值的有意义的位。

Gorilla 压缩算法将数据分为三种情况，如图 4 所示：

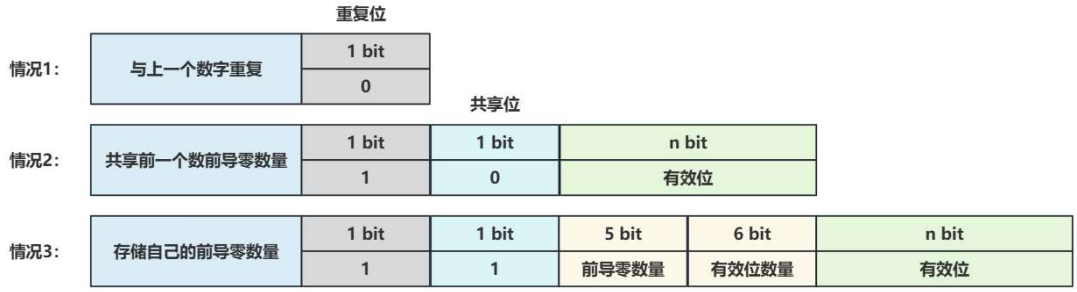

图 4. Goriila **编码**

Chimp[28]同样也是无损流压缩算法，是在 XOR 异或操作的时候将值与前面 128 个值进行比较，找到尾随零大于 6 位的，更好地找到与当前测量值相似的值，提高压缩效率。在 ELF[29]的浮点数尾数擦除方案中，提到根据有效值尾数对浮点数二进制尾数进行擦除，从而形成大量的尾随零，改进了异或结果的编码策略，进一步提高了压缩效率，如图 5 所示：

图 5. 浮点数二进制异或编码示例

ELF 由于低位对原数据大小影响较小，所以 ELF 也可以向上取整得到原数据。这种计算办法是通过计算有效位，最直观的方法是逐步减少尾数前缀数的位数，直到 Delta Δ 大于 10^(- α)，其中 α 是小数点后的位数。这种方法在最坏情况下需要进行 52 次检查，因为 IEEE 754 双精度浮点数的尾数部分有 52 位。即使使用二分查找优化，复杂度仍为 O(log252)，效率依然不够高。ELF 提出利用有效位的计算公式。

2.2.4.2 基于精度擦除的 ELF 算法改进

ELF 方案中存在三个判断，计算指数部分，确定浮点数整数部分有效位。计算精度，确定浮点数小数部分的位数保留位。然后保留。本文提出的优化算法是：计算浮点数减去精度，获得新的浮点数，然后计算二进制数据中共同的相似位，保留到原数据中的相同位后一位。

原浮点数据 IEEE 754 编码中，浮点数 x 后会接有很多位数，这些位数带来很多尾随有效位。浮点数 x 减去精度 a，形成新浮点数 y，那么 x 一定比 y 要大，在浮点数位中 x 一定比 y 先出现不一样的 0 或者 1，共同部分长度为 len 保留，x 除了共同部分保留外还需要额外保留 1 位，形成保留位 len。这个 len 代表 x 在损失 0.001 的精度范围内保留最大的位数，如表 3 所示：

表 3.以浮点数为例的二进制擦除示例

	浮点数	二进制（整数部分）	二进制（小数部分）
原始数据	113.119	1110001	00011110011101101100100010110100000111001010
减去精度后数据	113.118	1110001	00011110001101010011111101111100111011011001
擦除位数后的数据	113.1181640 6	1110001	00011110011101101~~1001000101101000011100101~~0

找到数据的一个合适的低位置为零，之后进行异或操作，可以产生更多尾随

零。表 3 的尾随零就从 1bit，提高到了 28bits，这样压缩率较直接进行异或有着明显提升。

压缩部分按浮点数进行二级压缩，首先进行 ELF 优化后的处理，尾随零擦除部分需要考虑一个因素：是否需要擦除。大部分的数据基本上都可以进行擦除，在这里为了简化，与擦除后的原数据增加判断：1.如果原数据绝对值小于 0.001，跳过 elf 擦除部分，用特殊形式编码；2.擦除前后如果原数据没有变化（原数据本身后面不会存在冗余的位数部分），那么标志位 1 代表未擦除，其余位标志位 0 代表擦除。

基于此提出精度感知的 ELF 算法改进：

算法 1: 基于精度感知的 ELF 压缩算法

输入: 输入浮点数序列 $F = \{f_1, f_2, ..., f_n\}$，精度阈值 ϵ

输出: 压缩后的二进制序列 B 及比特数据 M

过程 *ELF 精度压缩* (F, ϵ)
  对于 $f_i \in F$ **执行**
    $sign_i \leftarrow$ 符号 (f_i) // 提取符号位
    $\Delta \leftarrow sign_i \cdot \epsilon$ // 根据符号确定精度偏移
    $f_i' \leftarrow f_i - \Delta$ // 生成偏移值
    $b_1 \leftarrow$ 二进制表示 (f_i) // 原始二进制表示
    $b_2 \leftarrow$ 二进制表示 (f_i') // 偏移后二进制表示
    $c_{bits} \leftarrow$ 公共前缀 (b_1, b_2) // 提取公共前缀
    $s_{bit} \leftarrow$ 下一有效位 $(b_1, |c_{bits}|)$ // 保留一位有效位
    $t_{bits} \leftarrow$ 生成尾零 $(|b_1| - |c_{bits}| - 1)$ // 生成尾随零
    $b_i' \leftarrow c_{bits} \| s_{bit} \| t_{bits}$ // 构造压缩结果
    如果 $b_i' = b_1$ **那么**
      $M_i \leftarrow (0, |c_{bits}|, sign_i)$ // 未压缩标记
    如果结束
    否则
      $M_i \leftarrow (1, |c_{bits}|, sign_i)$ // 压缩标记
    如果结束
    $B \leftarrow B \cup \{b_i'\}$ // 添加到结果序列
  对于结束
  return (B, M)
过程结束
过程 *ELF 基于精度解压缩* (B, M, ϵ)
  对于 $i \leftarrow 1$ **to** $|B|$ **执行**
    如果 $M_i[0] = 0$ **那么**
      $f_i \leftarrow$ 浮点重建 (B_i) // 直接重建
    如果结束
    否则
      $f_i' \leftarrow$ 浮点重建 (B_i) // 重建中间值
      $f_i \leftarrow f_i' + M_i[2] \cdot \epsilon$ // 应用精度恢复
    如果结束
    $F \leftarrow F \cup \{f_i\}$ // 重建原始序列
  对于结束
  return F
过程结束

对比实验结果如表 4：

表 4.改进前后算法速度对比

浮点数个数	1,0000	10,0000	100,0000	1000,0000	1,0000,0000
ELF	2ms	3ms	5ms	15ms	127ms
ELF 改进	0ms（近似）	0ms	2ms	2ms	2ms

进行尾数擦除后，尾数擦除后都会大量尾随零，在批量数据压缩时，有效位和前导零位的数量都在一定区间范围内，那么可以直接在压缩数据前提前写如前导零或尾随零的最大占用位数，后续的编码压缩就不用再写入前导零或尾随零数量的表示了，而是直接写入前导零或尾随零数量和有效位，相较原来 Goriila、Chimp 的压缩编码上会进一步压缩。

实现方法是增加控制器，控制器作用：1.控制前导零和尾随零以及有效位的数据长度大小。取更小的数据长度标识窗口长度。2.如果控制器所确定的前导零大小或者尾随零大小显著改变，则判定为异常情况。存储异常完整数据，下一个数据与异常前进行比较和异常数据进行比较，如果与异常数据更为相近（差值后占位小），同样存储为异常数据，如果与异常前数据相近，恢复为正常数据。3.如果异常之后，新数据与异常数据较为接近，且控制器设定的窗口大小恢复原状且持续时间 T，那么异常数据编码取消，判定恢复为正常情况。

完整编码情况如下：

（1）符号位 xor，尾数位相减取绝对值，记录数据递增（0）和递减（1）

控制器确定前导零/尾随零的长度以及有效位的长度。（例如 8 位，控制器取 3）

（2）控制器中前导零和尾随零长度如果突然变化（例如从 8 位前导零变为 1 位前导零，变化波动差大于 3），判定为异常，正常编码为 0，异常编码为 1。异常编码中存储完整数据。

（3）异常之后，新数据与异常前数据和异常后数据进行比较。新数据与异常数据后较为接近且则控制器设定的窗口大小持续时间为 T，（例如 T=16 个数据），那么异常编码取消。新数据与异常前数据接近，则返回正常数据编码。

（4）通过控制器控制的前导零和尾随零、有效位长度大小，确定是存储前导零还是尾随零以及分配给多少位存储前导零和尾随零和有效位。

2.2.4.3 基于数据分类的压缩选择策略

在存储时，根据数据类型，分为浮点数编码压缩和时间戳压缩，在块上通过存储差值来高效存储时间戳。因此在这里重点讨论基于浮点数的压缩。

对于时间相邻的数据，其呈现出值波动的相似性这一显著特征。那么进行二进制编码压缩时，这种相似性导致前后数据相同或相近的情况频繁出现。针对这

一现象，可以利用 XOR 差分编码的方式。利用相邻时间戳的异或结果，配合前导零和尾随零进行压缩，能够充分挖掘数据间的相关性，从而实现高效压缩。

除了时间相邻数据外，电力数据还存在其他具有鲜明特征的数据类型。其中，异常数据表现出较大的波动幅度，其数据值变化缺乏明显的规律，这使得其在压缩处理时需要特殊对待。而具有完整波形的数据则呈现出重复的波形特征，这种周期性的重复信息为数据压缩提供了另外一种优化思路。鉴于这些不同类型数据的特性差异，在编码方案的选择过程中，传统的单一压缩算法难以满足多样化的数据压缩需求。因此，采用自适应的编码选择策略具有至关重要的意义。该策略旨在全面考量压缩比和压缩时间这两个关键性能指标，在众多编码类型中筛选出最适合特定数据类型的压缩算法，从而实现整体性能更优的数据压缩效果，提升数据处理系统的综合效率。

自适应算法选择是根据输入的数据特征和性能要求，精确数据最优压缩算法，从而保证不同类型数据在压缩过程中都能达到最佳的性能平衡。

稳定模式：当数据呈现稳定模式时，其数据值在一定范围内波动较小且具有相对稳定的变化趋势。在这种情况下，以 XOR 编码（XOR_ENCODING）为基础的基于精度的擦除 ELF 无损压缩的改进算法被确定为候选算法。这种编码可以依据数据使用调整所需要的压缩精度，能够有效捕捉数据间的相似性，通过异或操作对相同或相近的数据进行高效编码，从而减少数据冗余。

周期模式：对于具有周期模式的数据，其在固定的时间间隔或数据序列中呈现出明显的重复性特征。针对此类数据，字典编码（DICT_ENCODING）和 XOR 编码（XOR_ENCODING）被选为候选算法。字典编码可以利用周期数据中的重复模式建立字典，通过对重复模式的索引来实现高效压缩，大大减少了数据的存储空间。基于窗口的 XOR 编码则可以挖掘时间窗口内的数据相似性，通过异或操作对相似部分进行编码，与字典编码相结合能够更充分地利用周期数据的特性，提高压缩效果。

异常模式：异常模式的数据以其较大的波动幅度和不规则的变化为主要特征，并且这类数据的使用价值较高，因而需要时间开销更少的压缩数据读取算法。流式数据的压缩例如 Snappy、基于精度的 ELF 改进算法等都被确定为候选算法。

对于每个候选算法，按照以下公式计算归一化的指标。对于算法 A：

$$CRAj = \frac{Aj_{压缩比}}{\sum_{i\in候选算法集合} Ai_{压缩比}} \quad (1)$$

$$CTAj = \frac{Aj_{压缩时间}}{\sum_{i\in候选算法集合} Ai_{压缩时间}} \quad (2)$$

$$CPUAj = \frac{Aj_{CPU\ 开销}}{\sum_{i\in候选算法集合} Ai_{CPU\ 开销}} \quad (3)$$

$$MEMAj = \frac{Aj_{内存开销}}{\sum_{i\in候选算法集合} Ai_{内存开销}} \quad (4)$$

这里的 $Aj_{压缩比}$、$Aj_{压缩时间}$、$Aj_{CPU\ 开销}$、$Aj_{内存开销}$是算法 Aj 实际测量得到的压缩比、压缩时间、CPU 开销、内存开销，Ai 为使用其他候选算法计算得到的值，计算方法是相对评估，每个指标都是该算法在所有候选算法中的相对表现。针对每一个处于候选算法集合中的算法，执行评估和选择步骤。通过

$$ScoreA = P_{压缩比} \times CRA + P_{时间} \times CTA + P_{CPU\ 开销} \times CPUA + P_{内存开销} \times MEMA \quad (5)$$

$$P_{压缩比} + P_{时间} + P_{cpu\ 开销} + P_{内存开销} = 1 \quad (6)$$

计算每个候选算法的得分 ScoreA。根据数据特征处理应用场景以及对压缩性能的侧重要求，来确定压缩比权重、压缩解压缩时间权重、CPU 开销权重和内存开销权重权重和应为 1。

比较所有候选算法的得分，将各个候选算法计算得到的得分进行排序，得分最高的算法即在当前权重设定和数据处理场景下最优压缩算法。

2.2.5 实验整体框架

（1）数据分类以及特征编码

按照定义的数据类别和特性，将数据按照固定窗口进行划分，对每一个窗口的数据进行特征的识别和编码。

（2）浮点数压缩实验

对电力数据集运用基于精度调整的改进 ELF 浮点数压缩算法，与传统的数据库压缩算法，流式数据压缩算法进行性能比对，包括压缩比、压缩时间、CPU 性能和内存性能。

（3）数据块压缩存储

按照固定块大小划分存储块，对块内不同的编码数据运用自适应的压缩算法。分别与关系型数据库、时序数据库上的压缩性能进行比对。

（4）查询性能比对

与现有的数据库进行查询性能比对，查询类型包括聚合查询、范围查询、相似性查询，比对指标包括响应延迟、查询吞吐量、查询准确率。

（5）系统的性能比对与分析

通过与现有数据库存储效率、查询性能、系统扩展能力的比对进行优势分析，验证特征识别能力、查询灵活性与应用适应性。最后进行改进空间分析。

3. 已取得的进展

(1) 完成电力数据的前期调研分析，包括数据分类，电力数据应用场景分析。

(2) 实现电力数据特征识别工作。明确了数据特征类型和识别方法。

(3) 进行相关文献的调研和综述，从特征分段、索引、压缩等各角度都有了充分的方法应用了解。并结合数据特征基础，设计除了完整的存储方案和实现方法。

(4) 根据现有的浮点数压缩方法，提出基于精度上的浮点数压缩策略，以实现时间复杂度和效率上的改进。并结合时序特征设计自适应的编码方案和策略。

(5) 基于电力存储系统分块重新设计组织模型，对块的存储结合索引重新设计，以达到与性能优化的需求。

(6) 已完成 B+树的索引代码和浮点数编码压缩代码并做了性能对比，为方法的可行性奠定了实践基础。

4. 后续研究计划

2025 年 1 月完成系统的搭建，包括特征识别机制的完善，验证特征分类编码和分类的准确率和效率；数据块存储模型的实现以及索引与存储的结合，选定特定数据库模拟大规模电力数据存储测试。采用数据库查询工具完成各模块在数据库上对应的系统测试和性能测试，分析以获得结果验证。

2025 年 2 月总结实验和项目研究完成内容，撰写结题论文。分析系统整体性能，找出优化点；整理实验数据，撰写初步分析报告；并根据实验情况对查询

过滤、特征识别分类等算法做出调整。

5. 预期结果

5.1 理论创新成果

特征编码模型上取得模型的创新，包括电力特征识别、自适应编码机制、形式化定义体系。索引理论上实现多维索引、特征结构感知、动态维护策略和查询优化的创新。

5.2 技术创新成果

核心算法例如压缩算法，自适应特征感知压缩，精度控制的浮点数编码压缩和多模式匹配、高效检索的查询算法创新。系统架构上有存储效率、查询性能和系统扩展性的优化和提升。

5.3 应用价值

在电力数据业务支持包括故障实时分析、模式精确分析、趋势准确预测方面取得功能增强，为实时监控、故障诊断、连锁故障分析、风险评估等方面提供决策支持。

6. 主要参考文献

[1] 李毅松，杨琦，胡楠，何江. 基于 SG-CIM 的电力信息通信系统监控模型. 电力信息化,2012,10(10):35-39.

[2] 肖子达,朱立谷,冯东煜等.分布式数据库聚合计算性能优化[J].计算机应用,2017,37(05):1251-1256.

[3] Rad B, Song F, Jacob V, et al. Explainable anomaly detection on high-dimensional time series data[C]//Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems. 2021: 2-14.

[4] Lipčák P, Macak M, Rossi B. Big data platform for smart grids power consumption anomaly detection[C]//2019 federated conference on computer science and information systems (FedCSIS). IEEE, 2019: 771-780.

[5] Naqvi S N Z, Yfantidou S, Zimányi E. Time series databases and influxdb[J]. Studienarbeit, Université Libre de Bruxelles, 2017, 12.

[6] Wang J, Tian Y, Hu X, et al. Development of grinding intelligent monitoring and big data-driven decision making expert system towards high efficiency and low energy consumption: experimental approach[J]. Journal of Intelligent Manufacturing, 2024, 35(3): 1013-1035.

[7] Bao Y, Huang Z, Gong X, et al. Optimizing segmented trajectory data storage with HBase for improved spatio-temporal query efficiency[J]. International Journal of Digital Earth, 2023, 16(1): 1124-1143.

[8] Liu X, Wei Z, Yu W, et al. Khronos: A Real-Time Indexing Framework for Time Series Databases on Large-Scale Performance Monitoring Systems[C]//Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. 2023: 1607-1616.

[9] Zhang L, Alghamdi N, Eltabakh M Y, et al. TARDIS: Distributed indexing framework for big time series data[C]//2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 2019: 1202-1213.

[10] Dai H K, Su H C. Clustering Analyses of Two-Dimensional Space-Filling Curves: Hilbert and z-Order Curves[J]. SN Computer Science, 2022, 4(1): 8.

[11] Guttman A. R-trees: A dynamic index structure for spatial searching[C]//Proceedings of the 1984 ACM SIGMOD international conference on Management of data. 1984: 47-57.

[12] Zhang X, Zhang G. Improved KD tree high dimensional index algorithm based on location information[C]//2020 International Conference on Image, Video Processing and Artificial Intelligence. SPIE, 2020, 11584: 169-173.

[13] Nathan V, Ding J, Alizadeh M, et al. Learning multi-dimensional indexes[C]//Proceedings of the 2020 ACM SIGMOD international conference on management of data. 2020: 985-1000.

[14] Ding L L, Qiao B, Wang G, et al. An efficient quad-tree based index structure for cloud data management[C]//Web-Age Information Management: 12th International Conference, WAIM 2011, Wuhan, China, September 14-16, 2011. Proceedings 12. Springer Berlin Heidelberg, 2011: 238-250.

[15]Luo C, Lou J G, Lin Q, et al. Correlating events with time series for incident diagnosis[C]//Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014: 1583-1592.

[16]Roggen D, Cuspinera L P, Pombo G, et al. Limited-memory warping LCSS for real-time low-power pattern recognition in wireless nodes[C]//Wireless Sensor Networks: 12th European Conference, EWSN 2015, Porto, Portugal, February 9-11, 2015. Proceedings 12. Springer International Publishing, 2015: 151-167.

[17]Wan S, Zhao Y, Wang T, et al. Multi-dimensional data indexing and range query processing via Voronoi diagram for internet of things[J]. Future Generation Computer Systems, 2019, 91: 382-391.

[18]Yao Z, Zhang J, Li T, et al. A trajectory big data storage model incorporating partitioning and spatio-temporal multidimensional hierarchical organization[J]. ISPRS International Journal of Geo-Information, 2022, 11(12): 621.

[19]Iwana B K, Frinken V, Uchida S. DTW-NN: A novel neural network for time series recognition using dynamic alignment between inputs and weights[J]. Knowledge-Based Systems, 2020, 188: 104971.

[20]Keogh E, Chakrabarti K, Pazzani M, et al. Locally adaptive dimensionality reduction for indexing large time series databases[C]//Proceedings of the 2001 ACM SIGMOD international conference on Management of data. 2001: 151-162.

[21]Kondylakis H, Dayan N, Zoumpatianos K, et al. Coconut: sortable summarizations for scalable indexes over static and streaming data series[J]. The VLDB Journal, 2019, 28: 847-869.

[22]Sears R, Ramakrishnan R. bLSM: a general purpose log structured merge tree[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. 2012: 217-228.

[23]Wang Y, Wang P, Pei J, et al. A data-adaptive and dynamic segmentation index for whole matching on time series[J]. Proceedings of the VLDB Endowment, 2013, 6(10): 793-804.

[24]Chiarot G, Silvestri C. Time series compression survey[J]. ACM Computing Surveys, 2023, 55(10): 1-32.

[25]Zhang Y, Zhang F, Li H, et al. CompressStreamDB: fine-grained adaptive stream processing without decompression[C]//2023 IEEE 39th International Conference on Data Engineering (ICDE). IEEE, 2023: 408-42.

[26]J. Wu, P. Wang, N. Pan, C. Wang, W. Wang and J. Wang, "KV-Match: A Subsequence Matching Approach Supporting Normalization and Time Warping," 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 2019, pp. 866-877, doi: 10.1109/ICDE.2019.00082.

[27]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.

[28]Liakos P, Papakonstantinopoulou K, Kotidis Y. Chimp: efficient lossless floating point compression for time series databases[J]. Proceedings of the VLDB Endowment, 2022, 15(11): 3058-3070.

[29]Li, Ruiyuan, Zheng Li, Yi Wu, Chao Chen, Songtao Guo, Ming Zhang, and Yu Zheng. Erasing-based lossless compression method for streaming floating-point time series. Proceedings of the VLDB Endowment, 2023,1736-1776.

研　究　生　签　字 _____

指　导　教　师　签　字 _____

院(系、所)领导签字 _____

年　　月　　日