软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, 2022,33(3):950−967 [doi: 10.13328/j.cnki.jos.006445]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 基于键值存储的分布式时序相似性搜索方法*

俞自生[1,3], 李瑞远[2,3], 郭 阳[4], 蒋忠元[1], 鲍 捷[3], 郑 宇[3]

[1](西安电子科技大学 网络与信息安全学院, 陕西 西安 710126)

[2](重庆大学 计算机学院, 重庆 400044)

[3](北京京东智能城市大数据研究院, 北京 100176)

[4](北京航空航天大学 计算机学院, 北京 100191)

通信作者: 李瑞远, E-mail: liruiyuan@whu.edu.cn

**摘 要**: 时序相似性搜索是时序数据分析最基本的操作之一, 具有广泛的应用场景. 针对现有分布式算法无法应对维度增长、扫描范围过大和相似性计算耗时的问题, 提出一种面向键值存储的分布式时序相似性搜索方法 KV-Search. 首先对时序数据分块, 并设计其键值存入键值数据库, 解决了时序数据维度高且不断增长的问题; 其次, 基于切比雪夫距离计算其下界, 并利用键值范围扫描提前过滤无效数据, 减少了数据传输; 最后, 利用基于分块的时序表示计算距离下界, 避免了更高维度真实数据的计算, 加快了查询效率. 使用 HBase 实现了 KV-Search, 并利用真实的大规模数据集做了大量实验. 实验结果表明, KV-Search 算法在效率和扩展性方面均优于基准实验.

**关键词**: 时间序列; 相似性搜索; 键值存储; 剪枝过滤; 分布式查询

**中图法分类号**: TP311

中文引用格式: 俞自生, 李瑞远, 郭阳, 蒋忠元, 鲍捷, 郑宇. 基于键值存储的分布式时序相似性搜索方法. 软件学报, 2022, 33(3): 950–967. http://www.jos.org.cn/1000-9825/6445.htm

英文引用格式: Yu ZS, Li RY, Guo Y, Jiang ZY, Bao J, Zheng Y. Distributed Time Series Similarity Search Method Based on Key-value Data Stores. Ruan Jian Xue Bao/Journal of Software, 2022, 33(3): 950–967 (in Chinese). http://www.jos.org.cn/1000-9825/6445.htm

## Distributed Time Series Similarity Search Method Based on Key-value Data Stores

YU Zi-Sheng[1,3], LI Rui-Yuan[2,3], GUO Yang[4], JIANG Zhong-Yuan[1], BAO Jie[3], ZHENG Yu[3]

[1](School of Cyber Engineering, Xidian University, Xi'an 710126, China)

[2](College of Computer Science, Chongqing University, Chongqing 400044, China)

[3](JD Intelligent Cities Research, Beijing 100176, China)

[4](School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

**Abstract**: Time series similarity search is one of the most basic operations for temporal data analysis, which has various application scenarios. Existing distributed methods face the problems of dimension explosion, too large scan range, and time-consuming similarity calculation. To this end, this study proposes a distributed time series similarity search algorithm KV-Search. First, time series are segmented into blocks and stored in the key-value database, which solves the problem of high and growing dimension. Second, the lower bound is calculated based on Chebyshev distance, and the invalid data is filtered out in advance using key value range scanning, which reduce the data transmission and calculation overhead. Third, a block-based time series representation is used to calculate the lower bound of distance, which avoids the calculation of higher dimensional real data. KV-Search is implemented based on HBase, and a set of extensive experiments are conducted using both real and synthetic time series data. The results show that the proposed KV-Search is superior to benchmark experiment in efficiency and scalability.

* 基金项目: 国家重点研发计划(2019YFB2103201); 国家自然科学基金(61976168, 62076191, 61502375)
本文由"数据库系统新型技术"专题特约编辑李国良教授、于戈教授、杨俊教授和范举教授推荐.
收稿时间: 2021-06-30; 修改时间: 2021-07-31; 采用时间: 2021-09-13; jos 在线出版时间: 2021-10-21

**Key words**: time series; similarity search; key-value storage; pruning filtration; distributed query

时序数据, 即随时间推移而变化的数据, 在人们的日常生活中无处不在. 近 10 年来, 随着电子监控设备和智能穿戴设备的普及, 更是产生了海量的时序数据. 例如, 经过多年的发展, 火力发电行业的数字化程度已经达到了很高的水平. 以一台 60 万千瓦的中型火电机组为例, 其内置的上万个传感器, 每秒可以产生数万条实时监控数据.

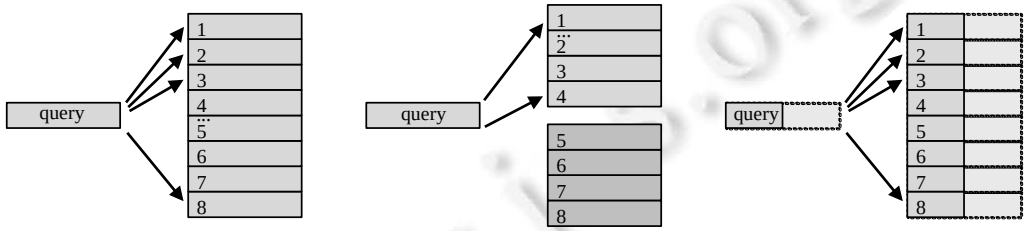
其中, 时序相似性查询, 即查询与给定序列最相似的 $k$ 个序列, 是最常用的时序分析算子之一[1−3], 可应用于推荐、聚类[4]和异常检测[5,6]等上层应用. 例如, 推荐算法中, 股票交易市场实时查询与用户持有股票走势最相似的若干股票用于推荐; 聚类应用中, 电力部门查询按天用电量相似的家庭用于聚类分析; 异常检测中, 火力发电行业查询与已知的异常状态序列相似的序列进行故障风险排查等.

在小规模数据下, 只需将给定时序与数据库中所有数据进行两两相似性计算后, 取最相似的 $k$ 条数据即可. 如图 1(a)所示, 查询序列需遍历数据库中所有序列. 显然, 时间复杂度为 $O(m)\times f(n)$. 其中, $m$ 为数据基数; $f(n)$为计算时序相似性的开销, 与相似度函数选取有关, $n$ 为时序维度. 然而, 在大数据背景下, 这种方法往往是不可行的, 主要挑战体现在如下 3 个方面.

- 第一, 时序数据基数大, 这就意味着传统的两两计算相似性哪怕是一种线性解决方案, 即仅需要扫描一遍数据库, 其耗时也是难以接受的.
- 第二, 时序数据维度高, 两条时序数据计算相似性的耗时也随之增加, 即 $f(n)$与 $n$ 一般成正比关系, 最终体现在总体查询时间的增加.
- 第三, 时序数据是不断产生的, 即其基数和维度均不断增加, 这给算法的设计带来了一定难度.

对此, 主要有两种处理方式.

- 第一, 如图 1(b)所示, 先对数据进行分区, 将相似的数据放入同一分区, 查询时只需扫描特定分区即可.其主要思想是降低 $O(m)$的复杂度, 即减少无效数据的扫描.
- 第二, 如图(c)所示, 先对数据进行降维, 查询时直接遍历计算降维后的数据; 同时, 局部敏感哈希方法能够以一定概率保证降维后的数据保持其在高维时的相似性关系不变. 其主要思想是降低 $f(n)$的复杂度, 即减少计算相似性的耗时.




(a) 线性扫描 (b) 分布式查询 (c) 局部敏感哈希

图 1 现有方法示意图

键值数据库作为 NoSQL 数据库的一种, 近年来发展迅速, 且因其海量的数据存储和高效的键值索引功能, 受到了广泛的关注. 但目前, 基于键值数据库的时序相似性查询研究很少. 对此, 针对现有技术存在的不足, 并结合键值数据库, 本文设计并实现了一种基于键值存储的分布式时序相似性搜索方法. 该工作是京东城市时空数据引擎 JUST (https://just.urban-computing.cn/)[7−9]的一部分, 命名为 KV-Search. 其中, JUST 是一款针对时空数据的数据引擎, 预置多种有效的时空挖掘算法, 提供集数据存储、查询、分析、可视化一体的解决方案, 能够高效地管理海量时空数据. KV-Search 是其时序数据子模块下的相似性查询算法的一部分. 本文的主要贡献如下:

(1) 提出了一种基于分块的时序表示方法, 将整条时序切分为多个时间分块, 并针对每个时间分块设计其键值以支持高效的查询, 解决了挑战一和挑战三.

(2) 提出了一种基于键值存储的分布式时序相似性搜索算法, 并提出了两种高效的剪枝策略, 即极值剪枝和分块剪枝, 解决了挑战二.

(3) 在大规模的真实数据集上进行了大量相关实验, 与基准实验对比, KV-Search 查询性能提升 10–100 倍, 充分验证了所提方法的高效性, 同时开源了所有的实验代码(https://github.com/ndbc-2021-kv-search/kv-search/).

本文第 1 节给出时序相似性搜索的相关概念及数学定义. 第 2 节介绍本文的相关工作. 第 3 节主要讲述如何对时序数据进行分块并存入键值数据库. 然后, 第 4 节给出基于上述键值存储的查询算法并提出两种高效的剪枝定理及其证明. 相关实验及其分析将会在第 5 节详细介绍. 第 6 节对该算法的扩展性进行简要分析. 最后对本文进行小结并展望未来的工作.

## 1 问题描述

下面给出一些基本定义.

表 1 列出了本文主要使用的数学符号及解释, 其详细定义如下.

表 1 数学符号及解释

| 符号 | 解释 |
|---|---|
| $X$ 或 $Y$ 或 $Q$ | 单条时序 |
| $\mathcal{D}$ | 时序数据集 |
| $D_c(X,Y)$ | 两条时序的切比雪夫距离 |
| $B_X$ | 时序的块表示 |
| $T_p$ | 切分时序的时间周期 |
| $X_B$ | 时序的块序列表示 |
| $D_b(B_1,B_2)$ | 块距离,切比雪夫距离下界 |
| $D_{bs}(X_B,Y_B)$ | 块序列距离,切比雪夫距离下界 |
| $\delta$ | 候选集与查询序列的距离最大值 |
| $D_{upper}(B_1,B_2)$ | 切比雪夫距离上界 |

**定义 1(时序, time series)**. 时序数据 $X$ 可表示成数值的序列 $X=\langle x_1,x_2,\ldots,x_n\rangle$, $x_i\in\mathbb{R}$, 并假设 $X$ 在每个时间戳 $t=1,2,\ldots,n$ 都有值, 其中, $|X|$为时序的长度, 即$|X|=n$. 令 $X^{i,j}$ 表示时间范围为 $t=i,i+1,\ldots,j$ 的时序 $X$ 的子序列 $X^{i,j}=\langle x^i,x^{i+1},\ldots,x^j\rangle$, 其中, $1\leqslant i\leqslant j\leqslant n$. 此外, 每条时序都有一个全局唯一的 $id$ 与其绑定. 值得注意的是, 即使时序长度不相等, 也可使用插值的方法进行补齐[10], 如全部补空值, 插值不参与计算即可. 本文为了方便起见, 假设所有时序维度相等且具有一个有效值.

令$\mathcal{D}=\langle X_1,X_2,\ldots,X_m\rangle$, $X_i\in\mathbb{R}^n$ 为时序数据集合, 且所有时序长度相等, 其中, $|\mathcal{D}|=m$ 为数据集的基数, $\mathcal{D}^{i,j}$ 表示时间范围在 $t=i,i+1,\ldots,j$ 的所有时序子序列集合, 即 $\mathcal{D}^{i,j}=\langle X_1^{i,j},X_2^{i,j},\ldots,X_m^{i,j}\rangle$.

**定义 2(切比雪夫距离, Chebyshev distance)**. 令 $X$ 和 $Y$ 为两条在 $t=1,2,\ldots,n$ 的时序数据, 则切比雪夫距离 $D_c$ 计算两时序对应位置上绝对值距离的最大值, 定义如下:

$$D_c(X,Y)=\lim_{k\to\infty}\left(\sum_{i=1}^{n}|x_i-y_i|^k\right)^{\frac{1}{k}}=\max_i|x_i-y_i|.$$

切比雪夫距离[11]旨在找出两两对应点绝对值距离的最大值, 是一种常用的相似性指标, 即距离越小, 相似性越大. 本文采用该距离衡量时序相似性, 原因有三.

- 第一, 切比雪夫距离计算简单, 适用于衡量两时序之间的波动情况. 例如, 股票市场利用该距离进行相似性查询, 可得到具有近似股价且波动范围较小的其他股票; 特征选择方法 C-LAS Relief 利用切比雪夫距离计算不同特征间的相似度[12].
- 第二, 计算切比雪夫距离时两序列对应点相互独立, 适用于分而治之的思想, 即可以先计算子序列之间的切比雪夫距离再聚合, 而不影响最终结果, 适合分布式计算.
- 第三, 切比雪夫距离涉及的是极值的计算, 即两两之间的最大值, 可将其映射为键值数据, 从而有效

地进行过滤.

综上, 本文使用切比雪夫距离, 下文无特殊说明均指该距离.

**定义 3(时序相似性搜索, time series similarity search)**. 给定时序数据集$\mathcal{D}$、查询时序 $Q$ 与其对应时间段 $T=[i,j]$, $1\leqslant i\leqslant j\leqslant n$ 以及近邻个数 $k$, 时序相似性搜索指基于切比雪夫距离在$\mathcal{D}$中找出与 $Q$ 最相似的 $k$ 个序列. 若令$\mathcal{R}$为时序相似搜索的结果集, 则$\mathcal{R}$应满足如下条件:

$$(\mathcal{R}\subseteq\mathcal{D}^{i,j})\wedge(|\mathcal{R}|=k)\wedge(\forall X\in\mathcal{R},Y\in\mathcal{D}^{i,j}-\mathcal{R},D_c(X,Q)\leqslant D_c(Y,Q)).$$

其中, 第 1 个条件$\mathcal{R}\subseteq\mathcal{D}^{i,j}$确保了查询结果满足时间范围约束 $T=[i,j]$; 第 2 个条件$|\mathcal{R}|=k$ 保证查询结果的大小为近邻个数, 一般情况下$|\mathcal{D}|>>k$, 故可保证该条件一定可以被满足; 第 3 个条件意味着不属于$\mathcal{R}$的序列, 其与查询序列 $Q$ 的距离一定大于等于$\mathcal{R}$中序列与 $Q$ 的距离, 进而保证结果为与 $Q$ 最相似的 $k$ 个序列.

如图 2 所示, 在 $T=[4,9]$的时间范围内查找时序 $Q$ 的最近邻, 即图中查询阴影覆盖的部分子序列. 其中, 切比雪夫距离已用箭头标出. 因为 $Y$ 与 $Q$ 的距离更近, 故 $Q$ 最相似时序为 $Y$.

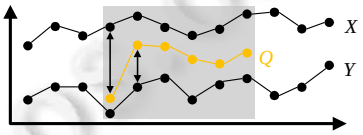



图 2 时序相似性搜索示意图

## 2 相关工作

### 2.1 分布式查询与时序索引

针对挑战一, 既然单机已经无法处理海量的时序数据, 那么基于分布式的思想, 可以先将相似的数据放在同一分区, 查询时仅需扫描给定序列所属分区即可[7,8,13−17], 如图 1(b)所示, 仅需在特定分区内查询即可获得结果, 其余分区可直接过滤. 例如, Yagoubi 等人[13]提出一种基于 iSAX 编码[18]的时序分区方法 DiSAX, 针对欧式距离的相似性度量, 对时序数据分区, 并创建本地和全局的二级索引, 将不同时序分发至不同分区进查询, 以支持分布式相似性查询.

为进一步加速相似性查询过程, 如何对数据进行索引就显得至关重要. 例如, Zoumpatianos 等人[19]提出了一种自适应的索引结构 ADS Index (adaptive data series index), 以更好地对时序数据进行索引, 从而加快查询; Peng 等人[20]提出了一种基于现代硬件并行化的分布式索引策略 ParIS (parallel index for sequences), 可完全消除磁盘驻留数据在索引构造过程中的 CPU 延迟; Linardi 等人[21]针对一般查询序列长度固定的问题, 对时序进行重新表示并建立索引, 提出了名为 ULISSE 的机制, 可应对不同长度的查询序列需求.

但上述方法没有面向切比雪夫距离有针对性地建立索引和分区查询, 并不能很好地支持本文所提的时序相似性搜索, 且现有工作中并没有基于键值数据库的分布式时序相似性搜索的研究.

### 2.2 局部敏感哈希

针对挑战二, 时序维度过高是导致查询耗时增加的原因之一, 那么对时序进行降维即可减少相似性计算开销. 一种经典的算法是局部敏感哈希 LSH (locality sensitive Hashing)[22−25], 其主要思想是: 在降维的同时, 以一定概率保证数据在高维时的相似性关系不变. 如图 1(c)所示, 对所有序列进行降维使其长度缩短, 从而加快了遍历时计算相似性的耗时. 例如, Alghamdi 等人[24]提出了一种单通哈希方法 SPS (single pass signature), 可以在加速降维过程的同时, 提高保存高维相似性的概率; Popivanov 等人[26]提出了一种基于小波变换的序列降维方法; 此外, 还包括主成分分析 PCA (principal component analysis)和奇异值分解 SVD (singular value decomposition)等矩阵方法也可对数据进行降维.

但值得注意的是, 局部敏感哈希的方法在降维的过程中, 原始时序的部分数值已经丢失, 导致降维后的时序失真, 无法还原序列之间真实的相似性, 且降维后的相似性关系也只是以一定概率保证. 所以这种方法

往往只能得到近似解, 不能得到精确解.

针对挑战三, 时序的基数和维度的不断增加, 给算法设计带来一定困难. 分布式的分区解决方案一定程度上能够应对这种情况. 例如, 文献[13]可以在新的数据到来后, 不断更新其建立的本地和全局的二级索引, 以达到支持后续查询的目的. 但值得注意的是, 这种索引的更新甚至重建是十分耗时的, 难以应对实时数据的查询需求.

## 3 基于键值的时序分块存储

### 3.1 时序的分块表示

时序数据是典型的海量且高维数据, 且其维度随时间的推移而不断增大, 故传统的按行存储, 即将序列的所有数值作为一个整体进行存储, 容易超过单机上限, 更是难以处理时序维度增加的情况. 例如, 在下一时刻 $t=n+1$ 时, 时序新增数值 $x_{n+1}$, 则基于整条时序的索引就需更新, 即挑战三. 因此, 本文采用分而治之的思想, 对时序进行切分存储[27], 以此应对时序维度高且不断增长的情况.

以下给出切分时序的一些概念定义.

**定义 4(块, block)**. 设时序 $X$, 定义块为 $B_X(X_{\max},X_{\min})$, 即使用时序的最大最小值表示时序整体. 这样设计的原因是显然的, 因为切比雪夫距离计算的是最大值, 而使用极值表示的块可以最大限度地保留原有信息, 并且对原数据进行了常量式地压缩.

**定义 5(块序列表示, block sequence representation)**. 设时序 $X$ 且$|X|=n$, 给定一个时间周期长度 $T_p\in\mathbb{N}^+$, 对 $X$ 划分子序列, 规则是 $x_i$ 的编号为$\lceil i/T_p\rceil$. 每个子序列使用其块进行表示. 最终的划分结果为

$$X_B=\langle B_1,B_2,...,B_{N_p}\rangle,$$

其中, 记 $B_i=(\max_i,\min_i)$, 称为块序列 $X_B$ 的索引下标为 $i$ 的分块, 共计分为 $N_p=\lceil n/T_p\rceil$块.

如图 3 所示, 时序 $X$ 被分为两块, 长度均为周期长度 $T_p=6$, 每个分块使用其包含的子序列的极值表示, 即 $B_1=(x_2,x_6)$和 $B_2=(x_{11},x_7)$. 注意: 前 $N_p-1$ 个块的长度都是周期长度 $T_p$, 但最后一个块可能因为时序长度$|X|$并不是 $T_p$ 的整数倍, 而出现不足 $T_p$ 个值的情况.

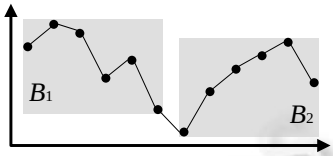



图 3 时序数据块序列表示示意图

**定义 6(块距离, block distance)**. 设 $B_1=(\max_1,\min_1)$和 $B_2=(\max_2,\min_2)$两个块, 则块距离 $D_b$ 定义如下:

$$D_b(B_1,B_2)=\max\begin{Bmatrix}\min\{|\max_1-\max_2|,|\max_1-\min_2|\},\\ \min\{|\min_1-\max_2|,|\min_1-\min_2|\},\\ \min\{|\max_2-\max_1|,|\max_2-\min_1|\},\\ \min\{|\min_2-\max_1|,|\min_2-\min_1|\}\end{Bmatrix}$$

不妨假设 $\max_1\geqslant\max_2$, 图 4 列举了 $B_1$ 和 $B_2$ 相对位置的 3 种情况, 块距离为同一直线单向箭头所指距离的最小值的最大值.

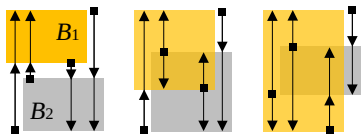



图 4 块距离示意图

**定义 7(块序列距离, block sequence distance)**. 设 $X_B$ 和 $Y_B$ 分别为等长时序 $X$ 和 $Y$ 在时间周期 $T_p$ 的分块表示, 则定义基于分块的块序列距离 $D_{bs}$ 如下:

$$D_{bs}(X_B,Y_B)=\max_{i=1,2,...,N_p} D_b(B_i^X,B_i^Y).$$

考虑到每个块仅使用极值表示, 其真实数值已被隐藏, 故无法通过块序列距离计算得到时序之间真实的切比雪夫距离. 但根据块序列距离, 可以计算得到切比雪夫距离的下界. 对此, 给出以下两个定理.

**定理 1(块距离下界定理)**. 块距离是其表示的真实序列的切比雪夫距离的下界.

证明: 设时序 $X=\langle x_1,x_2,\ldots,x_n\rangle$ 和 $Y=\langle y_1,y_2,\ldots,y_n\rangle$, 且 $B_X=(X_{\max},X_{\min})$ 和 $B_Y=(Y_{\max},Y_{\min})$ 为其块表示, 考虑两时序的极值情况, 则对于 $|X_{\max}-y_a|$, 其中, $y_a$ 为 $X_{\max}$ 在 $Y$ 的对应点, 且值一定在 $|X_{\max}-Y_{\max}|$ 和 $|X_{\max}-Y_{\min}|$ 之间, 故 $|X_{\max}-y_a|\geqslant \min\{|X_{\max}-Y_{\max}|,|X_{\max}-Y_{\min}|\}$. 同理可知, 其余极值点的绝对值距离的最小值, 即

$$D_c(X,Y)=\max_i|x_i-y_i|\geqslant\max\begin{Bmatrix}|X_{\max}-y_a|,\\|X_{\min}-y_b|,\\|Y_{\max}-x_c|,\\|Y_{\min}-x_d|\end{Bmatrix}\geqslant\max\begin{Bmatrix}\min\{|X_{\max}-Y_{\max}|,|X_{\max}-Y_{\min}|\},\\\min\{|X_{\min}-Y_{\max}|,|X_{\min}-Y_{\min}|\},\\\min\{|Y_{\max}-X_{\max}|,|Y_{\max}-X_{\min}|\},\\\min\{|Y_{\min}-X_{\max}|,|Y_{\min}-X_{\min}|\}\end{Bmatrix}=D_b(B_X,B_Y),$$

其中, $y_b$ 为 $X_{\min}$ 在 $Y$ 的对应点, $x_c$ 为 $Y_{\max}$ 在 $X$ 的对应点, $x_d$ 为 $Y_{\min}$ 在 $X$ 的对应点.

综上可得, $D_b(B_X,B_Y)$ 是 $D_c(X,Y)$ 的下界. 即对任意时序, 其块距离是切比雪夫距离的下界. □

**定理 2(块序列距离下界定理)**. 块序列距离是其表示的真实序列的切比雪夫距离的下界.

证明: 设时序 $X$ 和 $Y$, 令 $X_B$ 和 $Y_B$ 分别为 $X$ 和 $Y$ 在时间周期 $T_p$ 的块序列表示, 则

$$D_c(X,Y)=\max_i|x_i-y_i|=\max_{i=j*T_p,\,j=0,1,...,N_p-1}D_c(X^{i,i+T_p},Y^{i,i+T_p})\geqslant\max_{i=1,2,...,N_p}D_b(B_i^X,B_i^Y)=D_{bs}(X_B,Y_B).$$

综上可得, $D_{bs}(X_B,Y_B)$ 是 $D_c(X,Y)$ 的下界. 即对任一时序, 其块序列距离是切比雪夫距离的下界. □

### 3.2 基于键值的时序存储策略

HBase[28,29]是一个开源的面向列的非关系型分布式数据库, 目标是存储并处理海量数据, 并拥有高效的键值索引能力. 本文将时序数据进行切分后分别存入 HBase, 并通过精心设计的行键支持基于切比雪夫距离的时序相似性搜索, 下文将详细介绍. 尽管本文是基于 HBase 实现了 KV-Search, 但值得注意的是, 其他任何键值数据库都可以较容易实现本文所提出的算法.

HBase 采用键值对 KeyValue 的形式进行列存储, 其中, RowKey 为其行键, 标识唯一一行. HBase 检索数据有以下 3 种方式: 第一, 通过 Get 的方式, 指定单个 RowKey 值直接访问并获取唯一一条记录; 第二, 通过 Scan 的方式, 设置 RowKey 的范围 StartRow 和 StopRow, 扫描得到一批数据; 第三, 通过全表扫描的方式, 即直接扫描整张表中的所有数据. 因此, RowKey 的设计十分重要.

设时序 $X=\langle x_1,x_2,\ldots,x_n\rangle$ 和基于时间周期 $T_p$ 的块序列表示 $X_B=\langle B_1,B_2,...,B_{N_p}\rangle$, 称 $B_i$ 为时间分块, 其中, $i$ 为其索引下标, 则行键 RowKey 设计如下:

$$RowKey: i@B_i^{\max}@B_i^{\min}@id,$$

其中, "@"符号仅为分隔符, 并不表示实际意义且不会实际存储. 在 HBase 实际存储中, RowKey 为字节数组, $i$ 和 $id$ 为整型各占 4 个字节, $B_i^{\max}$ 和 $B_i^{\min}$ 为浮点型各占 8 个字节, 故该 RowKey 共计占用 4+8+8+4=24 字节.

其中, $i=1,2,\ldots,N_p$ 为时间分块下标, $B_i^{\max}$ 和 $B_i^{\min}$ 为时间分块 $B_i$ 的最大和最小值, $id$ 为时序 $X$ 的标识. 这里, $i$ 和 $id$ 的组合可以保证行键的全局唯一性, 因为 $id$ 可确定唯一一条时序记录, $i$ 可确定该时序的唯一一个时间分块. 注意, 这里没有使用全局的最大最小值 $X_{\max}$ 和 $X_{\min}$. 这是因为若该时序新增一些维度, 其极值就可能发生变化, 则 RowKey 也应随之更改. 而频繁增删 RowKey 会严重影响效率, 故采用时间分块 $B_i$ 的极值, 即一旦分块确定, 其极值与相应的 RowKey 也随之确定, 不会再发生变化, 这极大地提高了算法的鲁棒性和扩展性. 实际情况中, 时序读数一旦产生将不会改变, 故本文并不考虑时序数值更新的情况.

此外, 针对每个时间分块 $B_i$, 按时间周期 $T_p'$ 再次进行分块得到 $B_B'=\{B_1',B_2',...,B_{N_p'}'\}$, 称为基于时间分块的

值分块, 其目的是充分利用下文中的块序列剪枝定理对数据进行过滤. 注意: 因时间分块的长度为 $T_p$, 故一般 $T'_p < T_p$. 下文称 $T'_p$ 为分块时间周期. 综上, Value 的设计分为两部分: 一是分块 $B_i$ 的分块表示 $B_B^i$, 二是分块 $B_i$ 所代表的真实时序数据. 具体为将二者存入 HBase 同一列族的两个不同列中.

注意: 如上文所述, 最后一个时间分块 $B_{N_p}$ 可能会出现不满足时间周期 $T_p$, 此时并不将其存入 HBase, 而是缓存在内存队列中, 后续随着数据不断增加再次满足 $T_p$ 时, 再进行处理存入 HBase. 若期间查询序列涉及到 $B_{N_p}$ 及其所包含的序列, 直接在内存中获取参与计算即可.

如图 5 所示, 时序 $X=\langle x_1,x_2,\ldots,x_{12}\rangle$, 其长度 $n$=12 且唯一标识为 $id_1$, 令时间分块的时间周期为 $T_p$=6 和每个时间分块的值分块的时间周期为 $T'_p=3$, 则该条时序存入 HBase 共分两个步骤.

- 第 1 步, 分块. 即先对原始时序数据按 $T_p$ 分为若干时间分块, 再对每个时间分块按 $T'_p$ 进行值分块. 例如, 时序 $X$ 先被分为 $B_1$ 和 $B_2$ 两个时间分块, 其长度均为 6; 然后对时间分块 $B_1$ 和 $B_2$ 再次分块, 分为 $[B_1^1,B_1^2]$ 和 $[B_2^1,B_2^2]$, 即每个时间分块又被分为了两个更小的块.
- 第 2 步, 插入. 即针对每个时间分块, 计算键值并存入数据库. 例如, 按上述键值设计方案, $B_1$ 的行键 Key 为 $1@x_2@x_6@id_1$, 其中, "1"为 $B_1$ 的下标索引, $x_2$ 和 $x_6$ 分别为 $B_1$ 的最大和最小值, $id_1$ 为时序 $X$ 的唯一标识. 每行数据的值 Value 分为两部分: 第 1 部分为时间分块的值分块, 如 $B_1$ 的值分块为 $[B_1^1,B_1^2]$; 第 2 部分为时间分块所代表的原始数据, 如 $B_1$ 的原始数据为$\langle x_1,x_2,\ldots,x_6\rangle$.

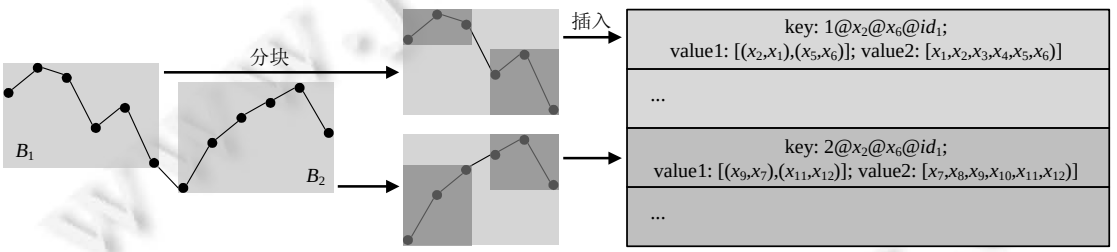



图 5 键值存储示意图

下面对时间周期 $T_p$ 的取值进行讨论. 由上可知, 时间周期 $T_p$ 决定子序列的长度, 故其取值范围为$[1,n]$, 其中, $n$ 为原始序列的长度. 考虑两种极端情况.

- 第一, 当 $T_p$=1 时, 即将时序 $X$ 的每个值都看作一个时间分块. 这样做的缺点有二: 首先是使得数据库的数据量暴增, 达到 $m\times n$ 条, 其中, $m$ 为原始时序的条数; 其次, 使得查询时下文中极值剪枝定理失去了作用, 因为相当于所有时序的所有值都被扫描了一遍, 等同于线性遍历, 严重影响效率.
- 第二, 当 $T_p=n$ 时, 即将时序 $X$ 整体看作一个时间分块, 等同于未对时序进行切分. 此时, 数据库的数据量最少为 $m$, 但这却也违背了切分时序的初衷, 即不断产生的时序数据需积累一定时间才能再次满足时间周期的条件后, 以键值的形式存入数据库, 也无法利用切比雪夫距离"分而治之"的计算思想, 对每条时序分布式计算其切比雪夫距离.

结合两种极端情况, 时间周期 $T_p$ 的取值不能太大也不能太小, 若时序数据动态产生, 则还需考虑数据的产生速率, 使其能够在合理时间内存储键值数据库以加快查询进程. 下文实验将会对不同时间周期的查询效率进行对比.

## 4 基于键值存储的相似性搜索算法

### 4.1 剪枝算法

为了尽量避免扫描无效数据, 减轻 IO 和内存开销, 基于上述时序存储方法, 提出以下两个剪枝定理.

**定理 3(极值剪枝定理)**. 设查询时序 $Q$ 和候选集所有序列与 $Q$ 的最大距离为$\delta$, 对于数据集$\mathcal{D}$中任意时序

$X$, 若$(X_{\min},X_{\max})\not\subset(Q_{\min}-\delta,Q_{\max}+\delta)$, 则 $X\notin\mathcal{R}$.

证明: 图 6(a)展示了极值剪枝的所对应的范围, 其证明具体分以下两种情况讨论.

1) 若 $X_{\min}\leqslant Q_{\min}-\delta$, 如图 6(b)所示, 不妨设 $X$ 最小点为 $x_1$, 并设 $Q$ 与其对应的点为 $q_1$, 则

$$D_c(X,Q)=\max_i|x_i-q_i|=\max_i\{q_1-x_1,...\}\geqslant q_1-x_1\geqslant Q_{\min}-x_1\geqslant\delta.$$

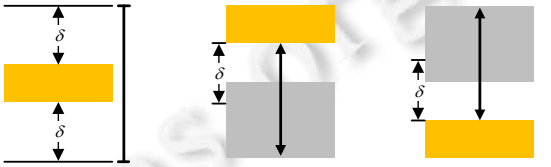
即 $X$ 的最小点对应的绝对距离一定大于等于$\delta$, 从而二者切比雪夫距离一定大于等于$\delta$.

2) 若 $X_{\max}\geqslant Q_{\max}+\delta$, 如图 6(c)所示, 不妨设 $X$ 最大点为 $x_1$, 并设 $Q$ 与其对应的点为 $q_1$, 则

$$D_c(X,Q)=\max_i|x_i-q_i|=\max_i\{x_1-q_1,...\}\geqslant x_1-q_1\geqslant x_1-Q_{\max}\geqslant\delta.$$

即 $X$ 的最大点对应的距离一定大于等于$\delta$, 从而二者切比雪夫距离一定大于等于$\delta$.

综合情况 1)和情况 2), 得证. □




(a) 极值剪枝 (b) 最小值情况 (c) 最大值情况

图 6 极值剪枝示意图

定理 3 在实际实现中可根据行键 RowKey 的范围扫描提前过滤无效数据, 又因为 $X_{\min}\leqslant X_{\max}$, 则只需保证 $X_{\max}\notin(Q_{\min}-\delta,Q_{\max}+\delta)$即可. 这样虽然扫描范围变大, 但是可以利用 HBase 的 RowKey 前缀匹配模式, 即仅匹配行键头部的特定几个字符, 可以大大加快搜索进程.

**定理 4(分块剪枝定理)**. 设查询时序 $Q$ 和候选集中最大距离为$\delta$, 对于数据集$\mathcal{D}$中任意时序 $X$, 设 $Q$ 和 $X$ 的块序列表示分别为 $Q_B$ 和 $X_B$, 若 $D_{bs}(X_B,Q_B)\geqslant\delta$, 则 $X\notin\mathcal{R}$.

证明: 由定理 2 得, 块序列距离是其真实切比雪夫距离的下界. 则

$$D_c(X,Q)\geqslant D_{bs}(X_B,Q_B)\geqslant\delta$$

即 $X$ 和 $Q$ 的真实切比雪夫距离一定大于等于$\delta$, 不可能在最终结果集内. 综上得证. □

定理 4 可以在不计算真实时序切比雪夫距离的情况下, 使用其块序列距离过滤无效数据. 考虑到块序列距离的计算开销远远小于其真实距离计算, 故这样的预计算是值得的. 一般来说, 块序列距离计算的时间复杂度为 $O(N_p)$, 而切比雪夫距离为 $O(n)$, 其中, $N_p$ 为分块个数, $n$ 为时序维度, 而对时序进行切分后, 往往 $N_p<<n$.

注意到该定理要求预知一个$\delta$值, 即已知候选集中的最大距离. 对此, 本文提出一种基于采样的$\delta$值预估方法, 如算法 1 所示. 为简便起见, 本文采用随机采样的方式, 因为这样可保持数据的原始分布.

**算法 1**. $\delta$值预估算法.

输入: 数据集$\mathcal{D}$、采样率$\alpha\in[0,1]$、查询序列的块表示 $B_Q=(Q_{\max},Q_{\min})$.

输出: 预估的最大距离$\delta$值.

1. 对$\mathcal{D}$按$\alpha$进行随机采样并保存样本序列的块进行表示, 即 $\mathcal{D}_{sample}=\{B_1,B_2,...,B_{\lfloor|\mathcal{D}|*\alpha\rfloor}\}$
2. 计算采样近邻个数 $k_{sample}=\lfloor k\times\alpha\rfloor+1$
3. $Dist_{sample}=\varnothing$
4. **for** $B_i$ in $\mathcal{D}_{sample}$ **do**
5. $Dist_{sample}\leftarrow Dist_{sample}\cup\{D_{upper}(B_Q,B_i)\}$, 其中, $D_{upper}(B_Q,B_i)=\max\{|Q_{\max}-B_i^{\max}|,|Q_{\min}-B_i^{\min}|\}$
6. **end for**
7. $\delta\leftarrow Dist_{sample}$ 中第 $k_{sample}$ 大的值
8. **return** $\delta$

下面对算法 1 进行一些解释.

其中, 由定理 1 关于块距离的讨论易得, $D_{upper}(B_Q,B_i)$是 $B_Q$ 和 $B_i$ 所表示的真实的切比雪夫距离的上界, 即 $D_{upper}(B_1,B_2)\geqslant D_c(X_1,X_2)$. 这里之所以没有使用真实的切比雪夫距离 $D_c$ 或块距离 $D_b$, 是出于两方面的考虑: 一方面是为了降低计算成本, 保存采样后时序的块表示而不是原始序列, 同时这也降低了采样的存储成本; 另一方面, 计算切比雪夫距离的上界而不是下界, 是为了使预估出的$\delta$值在原始数据中尽可能覆盖更多的数据, 从而减少漏解的可能, 因为较小的$\delta$值可能会出现最终结果个数小于 $k$ 的极端情况, 从而需要进行二次查询, 这是需要尽量避免的, 下文查询算法 2 中会详细介绍. 此外, 注意到算法 1 的第 2 行, 在数据量随采样而减少的同时, 近邻个数 $k$ 也同比例降低了. 一种直观的解释就是, 近邻序列也为原始数据集$\mathcal{D}$的子集, 那么对$\mathcal{D}$采样的同时, 也就相当于对近邻序列进行了采样, 那么 $k$ 值也应相应减少. 另外, 随着采样率的增加, 样本分布会越来越接近原始的数据分布, 预估的$\delta$值也会更加接近 $k$ 近邻真实的最大距离. 但显然的是, 样本的增加也会带来计算上的耗时增加, 从而最终反映在查询时间上. 因此, 一个合理的采样率是必要的, 下文也会对不同的采样率的影响进行对比实验.

### 4.2 相似性搜索

本节重点讲述利用上述键值存储和剪枝算法, 如何进行时序相似性近邻查询, 见算法 2.

**算法 2**. 分布式相似性搜索算法.

输入: 数据集$\mathcal{D}$、时间周期 $T_p$ 和 $T_p'$、查询时序 $Q=\langle q_i,q_{i+1},\ldots,q_j\rangle$、近邻个数 $k$.

输出: 时序 $Q$ 的 $k$ 个最近邻.

1. 根据算法 1 计算得到$\delta$值
2. 对 $Q$ 按 $T_p$ 切分得 $Q_s=\langle Q_1,Q_2,...,Q_{N_p}\rangle$, 其中, $Q_i=Q^{(i-1)*T_p+i\,j*T_p+i}$ //① 查询序列分块
3. 计算 $Q$ 的时间分块索引下标集合 $S=\{1,2,\ldots,N_p\}$
4. **for** $i$ in $S$ **do** //② 分布式查询
5. $PrefixQueryRange_i\leftarrow[i@(Q_{\min}-\delta),i@(Q_{\max}+\delta)]$
6. 在 HBase 中进行键值前缀扫描 $Scan(PrefixQueryRange_i)$, 得到结果集 $R1_i$ //定理 3
7. 对 $Q_i$ 按 $T_p'$ 切分得 $Q_B^i$
8. $R2_i\leftarrow\varnothing$
9. **for** $r1$ in $R1_i$ **do**
10. **if** $D_{bc}(Q_B^i,r1.block)\leqslant\delta$ **then** //定理 4
11. $R2_i\leftarrow R2_i\cup\{r1\}$
12. **end for**
13. **end for**
14. 合并结果集 $R2=R2_1\cup R2_2\cup\ldots\cup R2_{N_p}$ //③ 结果合并
15. $R2\leftarrow filter(R2,count(r2.id)==N_p)$ //序列必须在所有时间分块的查询结果中才被保留
16. $\forall r2\in R2$, 计算 $D_c(Q,r2.seq)$, 其中, $r2.seq$ 为原始序列, 并按此距离进行升序排序, 取 top-$k$ 的结果, 记为$\mathcal{R}_k$
17. **if** $|\mathcal{R}_k|<k$ **then**
18. 计算$\mathcal{R}_k$与 $Q$ 的最大切比雪夫距离, 记为$\delta_k$
19. 令$\delta\leftarrow\max(\delta_k,2\times\delta)$, 重复第 4 步
20. **else then**
21. **return** $\mathcal{R}_k$
22. **end if**

下面对算法 2 进行一些解释.

第 1 行–第 3 行主要是一些准备工作, 如预估$\delta$值和对查询序列 $Q$ 分块等. 第 4 行–第 13 行针对查询序列 $Q$ 的每个时间分块 $Q_i$, 先是利用了极值剪枝定理构造其行键的前缀扫描区间, 缩小了数据库的查询范围; 再是利用分块剪枝定理进一步过滤, 有效地避免了无效数据真实距离的计算. 注意, 这里每个时间分块内的查询和计算是分布式并行进行的, 如每个时间分块 $Q_i$ 都可以单独起一个进程对数据库进行查询, 分块剪枝的逻辑也可以在扫描数据库记录的同时进行判断. 第 14 行对各个时间分块的查询结果集进行合并. 注意, 一条时序 $X$ 若属于最终结果集$\mathcal{R}_k$, 则其每一个时间分块都应被查询到, 即对应算法第 15 行的 *filter* 函数, 其中, $r2.id$ 表示该行记录所存储的时序的唯一标识 *id*. 第 16 行对过滤后的数据计算其与查询序列 $Q$ 的真实切比雪夫距离,并升序排序后取 top-$k$ 作为最终的结果集$\mathcal{R}_k$, 其中, $r2.seq$ 表示该行记录所存储的原始时序值. 最后, 算法第 17 行–第 22 行处理了当预估的$\delta$过小, 查询结果不足 $k$ 的情况, 即给$\delta$赋一个更大的值并进行相同操作的二次查询. 注意, 此时可不再扫描之前的查询过的范围, 直接获取上次扫描结果进行迭代. 事实上, 预估值往往都比较大, 故通常不会触发二次扫描.

算法 2 的查询过程如图 7 所示, 主要分 3 个步骤: 查询序列分块、分布式查询和结果集合并.

- 第 1 步, 对查询序列进行分块. 图中对长度为 12 的序列按时间周期为 6 进行分块, 分为 $B_1$=($x_2$,$x_6$)和 $B_2$=($x_{11}$,$x_7$)两块, 其中, $x_2$ 和 $x_6$ 分别为 $B_1$ 的最大最小值, $x_{11}$ 和 $x_7$ 分别为 $B_2$ 的最大最小值.
- 第 2 步, 对于每个时间分块, 根据极值剪枝定理生成键前缀范围并在键值数据库中并行查询. 例如, 对于图中 $B_1$, 生成键值扫描范围[1@$x_6$–$\delta$,1@$x_2$@+$\delta$], 其中, “1”代表时间分块下标“1”, 之后在数据库中查询, 收集查询结果.
- 第 3 步, 合并各个时间分块的查询结果, 得到最终结果 $k$ 近邻. 注意, 合并的规则是: 当且仅当某序列 *id* 在所有时间分块的查询结果中都出现才进行记录, 否则丢弃. 例如, 序列 $id_2$ 在两个查询结果中均出现, 故被记录并拼接还原为原先的序列; 而序 $id_1$ 仅在 $B_1$ 的查询结果中出现, 序列 $id_4$ 仅在 $B_2$ 的查询结果中出现, 故丢弃.

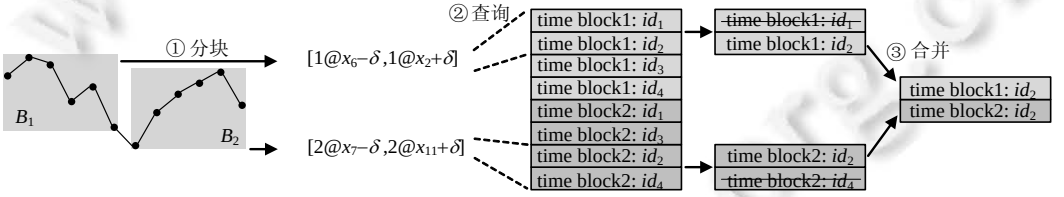



图 7 查询算法示意图

为验证该算法正确性, 给出以下定理并进行证明.

**定理 5(正确性定理)**. 算法 2 能够正确地查询出结果.

证明: 算法 2 在第 4 行–第 13 行利用了极值剪枝和分块剪枝两个定理, 在不同的时间分块上并行过滤无效的数据, 其正确性如上述证明. 因此, 可以保证合并后的中间结果集中一定包含查询序列 $Q$ 的 $k$ 近邻, 故第 16 行按与 $Q$ 的相似度排序取 top-$k$ 后, 即可得到最终的结果集$\mathcal{R}_k$.

但需要注意的是, 可能会出现预估的$\delta$过小, 从而导致 $k$ 近邻在剪枝时被过滤, 表现为合并后的中间结果集基数不足 $k$ 个. 为应对这种情况, 算法第 17 行–第 22 行判断了结果集的基数, 并当基数不足时, 给$\delta$赋予一个更大的值并再次查询, 直至查询到 $k$ 个为止. 综上得证. □

另外值得一提的是, 算法 2 的第 17 行–第 22 行中, $\delta$值看似是在搜索过程中不断收敛的, 其实不然, 因为在算法 1 的$\delta$值预估中采用的是切比雪夫距离上界, 故在样本和真实数据分布一致基础上, 其计算得到的$\delta$值是比真实的 $k$ 近邻的最大切比雪夫距离大的, 此时$\delta$值参与至算法 2 中仅需扫描一次即可得到最终的 $k$ 近邻结果, 算法如此设计是为了避免极端情况下漏解情况的发生.

算法 2 的复杂度分析如下.

为方便分析, 简化一下条件, 不妨时间周期 $T_p$=$n$, 即每条时序仅被分为一个时间分块, 此时数据库中的

数据总行数为$|\mathcal{D}_{\text{HBase}}|=|\mathcal{D}|=m$. 最好的情况下, $\delta$值预估的值正好为真实的查询序列与其近邻的最大距离值, 此时生成的前缀扫描范围可以覆盖所有近邻序列. 但由于定理 1 的块距离是真实切比雪夫距离的下界, 所以覆盖数据的总条数$|\mathcal{D}_{\text{HBase-cover}}|$一般会比近邻个数 $k$ 大, 即$|\mathcal{D}_{\text{HBase-cover}}|\geqslant k$, 当且仅当所有序列的块距离正好等于切比雪夫距离时等号成立. 最坏情况下, 预估的$\delta$值非常大, 使得生成的前缀扫描范围覆盖了数据库中的所有数据. 此时算法近乎失效, 即所有数据都需被遍历参与计算. 注意, 这里并没有考虑分块剪枝定理. 当时间周期 $T_p<n$ 时, 时序被切分为多个时间分块, 每个时间分块相当于一个独立的子序列, 其上的查询复杂度分析同上. 实际情况中, 查询的时间复杂度往往介于上述两种极端情况之间, 且因算法 1 预估的$\delta$值能够较好地以最小程度覆盖近邻序列, 故真实的查询效率会偏向于最好的情况. 综上, 该搜索算法的时间复杂度最好为 O($k$), 最坏为 O($m$), 其中, $k$ 为近邻个数, $m$ 为时序总条数. 一般情况下会偏向于 O($k$).

算法 2 的分布式特性分析如下.

该查询算法的分布式特性体现在第 4 行–第 13 行, 主要分为分布式查询与分布式剪枝两部分. 注意到定理 3 的极值剪枝在实际使用中是面向各个时间分块的, 即每个时间分块可单独使用极值剪枝, 而又因每条序列的行键最前缀为时间分块的索引下标, 故在数据库中, 不同下标的时间分块是被分在不同分区内的, 故算法第 4 行是可并行查询的且不会相互影响. 但定理 4 的分块剪枝是针对每一行, 判断该行数据是否可被过滤, 故并不具备分布式的能力. 综上, 算法 2 的分布式思想在于: 将整条序列 $Q$ 的查询切分为多个时间分块 $Q_B$ 上的查询子任务, 每个子查询可单独使用剪枝定理进行过滤, 且所有子任务可并行进行互不干扰, 最后将子查询结果进行合并处理.

## 5 实验与分析

### 5.1 实验数据与环境

本文实验环境为 5 台服务器组成的分布式集群, 单台计算资源为 10 核 CPU 和 10 GB 内存, 大数据组件为 Hadoop 2.7.3, HBase 1.4.9 和 Spark 2.3.3. 本文使用真实数据集 TS (https://archive.ics.uci.edu/ml/datasets/Condition+monitoring+of+hydraulic+systems)和 Traffic (https://pems.dot.ca.gov)根据高斯模型对原数据进行扰动而生成的仿真数据集 TS-F 和 Traffic-F, 其详细信息见表 2.其中, 数据量大小即为$|\mathcal{D}|$, 时序维度即为$|X|$, 数据大小为原始文本数据的大小. 对 TS-F 来说, 共有 185 220 个液压实验台, 每个实验台从第 1 秒到第 6 060 秒均有值, 值范围为 30–70; 对 Traffic-F 数据集来说, 共有 105 264 条公路, 每条公路从第 1 小时到第 18 102 小时均有值, 值范围为 0–1. 在基于切比雪夫距离的时序相似性搜索场景下, 对TS-F来说, 可查找出与某实验台温度数据相似的其他实验台, 且总体上每秒温度相差最小; 对 Traffic-F 来说, 可查找出与某公路占用率数据相似的其他公路, 且总体上每小时占用率相近, 以便于后续分析.

表 2 实验数据集

| 数据集 | 数据量 | 时序维度 | 数据大小(GB) | 数据范围 | 数据描述 |
|---|---|---|---|---|---|
| TS-F | 185 220 | 6 060 | 9.3 | [30,70] | 液压实验台每秒传感器温度, 单位: °C |
| Traffic-F | 105 264 | 18 102 | 13.1 | [0,1] | 旧金山每小时高速公路占用率的情况 |

实验参数见表 3, 其中, 加粗加下划线为默认参数. 下文中如无特殊说明, 均采用默认参数. 每个实验都会重复运行 10 次后取平均值作为最终结果. 查询序列 $Q$ 默认从$\mathcal{D}$中随机选取, 且其时间范围$[i,j]$默认为$[1,|Q|+1]$, 即从头开始截取长度为$|Q|$的子序列作为查询序列.

考虑到文献[23,24]得到的是近似解而非精确解, 文献[13]提出的 DiSAX 算法并不支持切比雪夫距离, 且暂未发现针对切比雪夫距离设计的分布式查询算法, 故本文设计以下 3 种基准实验进行对比.

(1) Spark-Scan. 该方法基于 Spark 对数据进行线性扫描, 具体做法如下: 先对全量数据按与查询序列的距离排序, 再取距离最小的 个序列作为最终的近邻结果返回. Spark-Scan 的时间复杂度为 O($m$), 其中, $m$ 为数据基数.

(2) Spark-Filter. 该方法基于 Spark, 并利用极值剪枝定理进行过滤, 具体做法如下: 数据预处理时, 保

存每条时序的最大值和最小值作为该时序的块表示, 形式如($B_X$,$X$), 其中, $X$ 为真实序列, $B_X$ 为该序列的块表示. 查询时, 先根据算法 1 预估一个$\delta$值, 然后线性扫描遍历每条序列, 利用定理 1 的块距离下界定理进行过滤, 即先计算该序列与查询序列的块距离 $D_b(B_X,B_Q)$, 若该值大于$\delta$, 则无须计算其真实的切比雪夫距离, 直接过滤该条数据. 与算法 2 相似, 若最终保留的数据条数不足 $k$ 个, 则增大$\delta$值再次扫描计算. 注意, 此时无须计算已经计算出真实距离的序列. 最后对数据按距离进行排序, 取距离最小的 $k$ 条数据作为结果返回即可. Spark-Filter 的时间复杂度为 O($m$), 其中, $m$ 为数据基数. 但由于上述的过滤效果, 查询时间理论上比 Spark-Scan 小.

(3) KV-Scan. 该方法基于 HBase 对数据进行线性扫描, 具体做法如下: 在原先时序分块键值存储的基础上, 不再使用极值剪枝定理和分块剪枝定理进行过滤, 直接进行全表扫描, 即先计算时间分块的切比雪夫距离, 再对每个时间分块的结果进行聚合, 最后按距离排序后取最小的 个序列作为结果返回. KV-Scan 的时间复杂度为 O($m\times N_p$), 其中, $m\times N_p$ 为 HBase 中数据总行数, 即每条时序被分为 $N_p$ 个时间分块, 每个时间分块单独存为一行.

表 3 实验参数

| 参数 | 取值 |
|---|---|
| $\mathcal{D}$ | [**TS-F**,Traffic-F] |
| $T_p$ | [**1000**,2000,3000,4000,5000,6000] |
| $T_p'$ | [**100**,200,300,400,500,600,700,800,900,1000] |
| $\alpha$ | [**1%**,5%,10%,15%,20%] |
| $k$ | [**100**,200,500,1000,1500,2000,3000,4000,5000,10000] |
| $\|Q\|$ | [1,10,100,1000,2000,4000,**6000**] |

### 5.2 实验结果与分析

#### 5.2.1 基准对比实验

为了展示本文所提出的 KV-Search 算法的性能, 将其与上述的基准算法进行对比, 结果如图 8 所示. 可以得到以下结论.

- 第一, KV-Search 性能平均比 Spark-Scan 快 20 倍左右, 比 Spark-Filter 快 10 倍左右, 比 KV-Scan 更是快了两个数量级. 这充分证明了本文所提算法的高效性.
- 第二, Spark-Scan 比 KV-Scan 快. 这是因为 Spark 是基于内存的操作, 而 HBase 需要将数据从磁盘取出并解析键值对, 涉及了大量的 IO 开销, 并且后者的数据扫描行数是前者的 $N_p$ 倍. 详情见上述对比实验分析.
- 第三, Spark-Filter 比 Spark-Scan 快近 1 倍. 这说明至少一半数据在计算块距离后就被提前过滤, 从而避免了更加耗时的真实的切比雪夫距离的计算, 最终体现在查询耗时上.
- 第四, KV-Search 比 KV-Scan 快. 这说明 KV-Search 在将数据加载至内存之前就已经过滤了大量的无效数据, 从而加快了整体的查询进程, 进而体现了极值剪枝定理和分块剪枝定理的高效性.

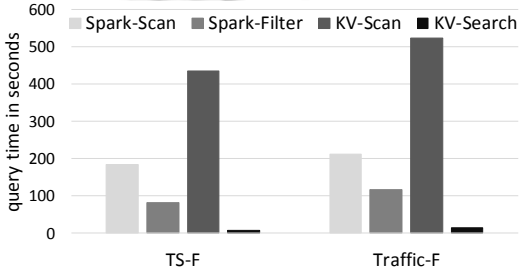



图 8 两个数据集的查询时间对比

5.2.2 不同的查询近邻个数 $k$

本节对比了查询不同近邻个数 $k$ 的运行时间，结果如图 9 所示. 可以得到以下结论.

- 第一，查询时间大体上随 $k$ 值的增加而增加. 这也是显然的，因为更大的 $k$ 值意味着需查询的近邻个数增加，所计算得到的$\delta$值也必然更大，从而覆盖了数据库中的更多行，参与计算的数据条数增大，计算耗时也随之增加.
- 第二，查询时间与预估的$\delta$值成正比关系. 即$\delta$值越大，查询时间越长，分析如上.
- 第三，相近的$\delta$值所对应的查询时间也相近. 如当 $k$=100, 200, 500 时，计算得到的$\delta$值基本相同，查询时间也基本相同，分析同样如上. 这反过来也告诉我们，那些较小的 $k$ 值的$\delta$值预估是有很大优化空间的.

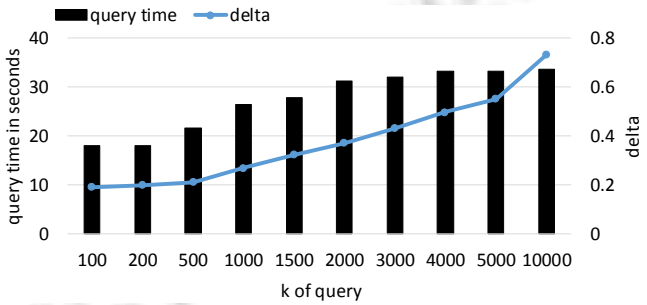



图 9 不同查询 $k$ 值的查询时间对比

5.2.3 不同的查询序列维度|$Q$|

该节对比了不同维度的查询序列对运行时间的影响，结果如图 10 所示. 可以看到，查询时间大体上随序列维度的增加而增大. 这是因为时序数据是分块存储的，一条时序被分为多个时间分块，每条时序的每个时间分块都是一条单独存在于数据库的行数据，被存储在与其索引下标对应的分区内，所以当查询序列维度|$Q$|增大时，想要查询某时序的所有值，就可能会跨越多个时间分块，从而需访问数据库的多个分区，进而而增大了扫描的数据条数，最终体现在查询耗时增加上.

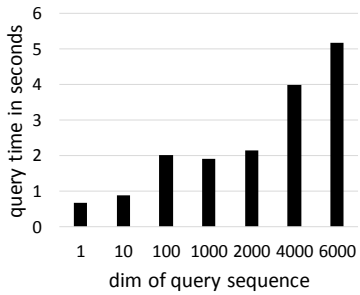



图 10 不同维度序列的查询时间对比

5.2.4 不同的时间周期 $T_p$

该节对比了不同时间周期 $T_p$ 对应的查询时间，结果如图 11 所示. 时间周期 $T_p$ 是时序分块的重要参数，时间分块数量 $N_p$ 与其成反比，即 $N_p=\lceil n/T_p \rceil$，其中，$n$ 为时序维度. 所以，随着时间周期 $T_p$ 的增加，时序切分的时间分块数量 $N_p$ 减少，即时序被存储在了更少的分区内，故要想获取某时序的全部数据，所需访问的分区数目也就更少，从而加快了查询. 值得注意的是，查询时间并非随着时间周期 $T_p$ 的增加而线性减少. 这是因为每个分区内的查询都是并行的，以空间换时间，一定程度上抵消了查询分区数增加的影响.

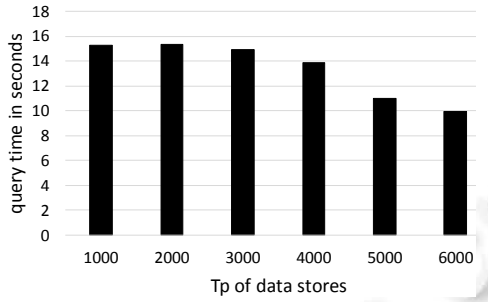



图 11　不同时间周期的影响

5.2.5　不同的分块时间周期 $T_p'$

该节对比了不同分块时间周期 $T_p'$ 对应的查询时间, 结果如图 12 所示. 分块时间周期 $T_p'$ 是指在将原始序列 $X$ 按时间周期 $T_p$ 切分后, 再对其每个子序列再次按 $T_p'$ 进行切分. 其目的是利用定理 4 的分块剪枝定理, 尽量避免真实的切比雪夫距离的计算. 不同的分块时间周期 $T_p'$ 会直接影响分块剪枝的效率: 理论上, $T_p'$ 越小, 其计算出的块序列距离就越接近真实的切比雪夫距离, 分块剪枝定理效果就更好. 当然, 其计算耗时也会增加. 关于块序列距离的计算复杂度的讨论见定理 4.

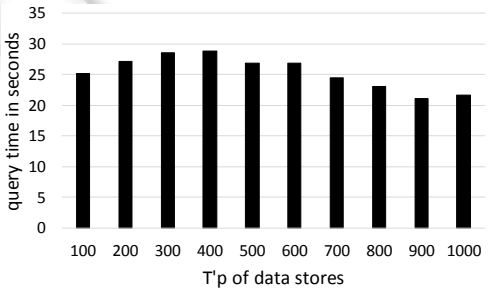



图 12　不同分块时间周期的影响

所以, 随分块时间周期 $T_p'$ 的增加, 分块剪枝效果减弱; 但同时, 块序列距离的计算复杂度减少. 二者在查询过程中共同作用, 最终使图 12 呈现先上升后下降的趋势, 而并非单调地上升或下降. 如图 12 所示, 当 $T_p'=400$ ,即分块时间周期大约为其表示的真实子序列长度 $T_p$ 的一半时, 耗时最长; 当 $T_p'<400$ 时, 分块剪枝的效果加强, 抵消了块序列距离计算的耗时, 总体耗时减少; 当 $T_p'>400$ 时, 块序列距离的计算复杂度降低, 抵消了分块剪枝的效果减弱带来的影响, 总体耗时减少. 注意, 这里使用的是默认参数 $T_p$=1000.

5.2.6　不同的采样率 $\alpha$

该节对比了采样率 $\alpha$ 对查询效率的影响, 结果如图 13 所示.

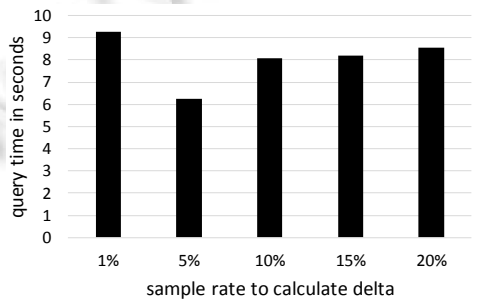



图 13　不同采样率的影响

可以看到, 查询时间随着采样率$\alpha$的增长, 呈现先下降后增长的趋势. 这是因为采样率的高低会直接影响采集的样本是否能反映查询序列与其近邻真实的最大距离, 而$\delta$值的计算是基于采样数据的, 所以采样率越高, 样本数据就更能反映原始数据的真实分布, 其计算得出的$\delta$值也就越准确, 即更能精确近似出查询序列真实的 $k$ 个近邻的最大距离, 进而可以过滤更多的无效数据, 这就产生了图中"先降"的趋势. 但值得注意的是, 随着采样率$\alpha$的不断增加, 样本数据随之增加, 则$\delta$值的计算成本变高, 反而增大了最终的查询时间, 这就产生了"后升"的趋势.

5.2.7 是否使用分块剪枝定理

为了进一步验证分块剪枝定理的作用, 本文将基准实验 Spark-Scan 与分块剪枝定理相结合, 具体做法如下: 先计算所有时序数据的分块表示并保存; 然后在遍历每条时序数据时, 不再直接计算其切比雪夫距离, 而是先使用其分块表示计算分块距离, 利用定理 4 的分块剪枝定理进行过滤. 其中, Block Filter 表示使用了分块剪枝定理, No Block Filter 表示没有使用, 其结果如图 14 所示.

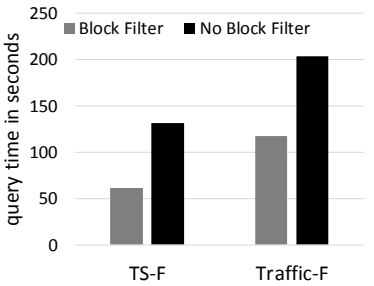



图 14 分块剪枝定理的影响

可以看到, 在全量数据扫描计算的情况下, 采用分块剪枝定理使得查询时间缩短了将近一半. 这说明至少一半以上的数据在未进行真实的切比雪夫距离计算前就已经被提前过滤了. 这充分表明了分块剪枝定理的高效性, 哪怕需额外保存时序的分块表示并计算其与查询序列的分块距离, 这种代价也是值得的.

## 6 扩展性分析

本文所提出的时序键值存储与查询框架易于扩展性, 大致可分为两方面: 第一, 该框架易于扩展满足其他查询需求; 第二, 该框架易于扩展满足其他相似性度量指标. 详细分析如下.

### 6.1 查询需求扩展

6.1.1 Dist-Query

Dist-Query, 即给定距离阈值$\varepsilon$, 查询出与序列 $Q$ 距离范围不超过$\varepsilon$的所有序列. 对应查询算法中, 只需令$\delta\leftarrow\varepsilon$, 其余步骤不变. 即将该距离阈值看作结果集与查询序列的最大距离.

6.1.2 Dist-KNN-Query

Dist-KNN-Query, 即给定距离阈值$\varepsilon$和近邻个数$k$, 查询出不超过$\varepsilon$的$k$近邻. 流程与 Dist-Query 相近, 仅需最后对结果排序取 top-$k$ 即可. 优化点可使用算法 1 预估$\delta$, 并令$\delta\leftarrow\min(\delta,\varepsilon)$. 这是因为若给定的$\varepsilon$很大而 $k$ 很小, 那么预估出的$\delta<<\varepsilon$, 直接使用$\varepsilon$构造前缀扫描范围只是徒增无效的数据计算. 取二者的最小值是因为, 若$\varepsilon<\delta$, 则预估的$\delta$值没有意义, 因为最终结果的距离均不能大于$\delta$.

### 6.2 相似性度量指标扩展

6.2.1 欧式距离

欧式距离定义为 $D_e(X,Y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_n-y_n)^2}$. 分块存储上, 可以令 $T_p$=1, 即将时序的每个值都作为一个时间分块进行存储, 即 $X_B$=⟨$B_1$,$B_2$,…,$B_n$⟩且 $B_i$=($x_i$,$x_i$), 行键 $RowKey=i@x_i@x_i@id$. 查询算法上, 可将距离函数替换为欧式距离, 利用算法 1 预估一个$\delta$值, 构造行键扫描范围[$i@x_i-\delta$,$i@x_i+\delta$], 其余步骤不

变. 因 $|x_i-y_i|=\sqrt{(x_i-y_i)^2}\leqslant D_e(X,Y)\leqslant\delta$，故相当于将距离$\delta$约束到序列的每一个值上.

显然，这样的做法存在严重的效率问题: 首先，将序列切分为单个值直接使数据库中的数据条数激增，即$|\mathcal{D}_{\text{HBase}}|=m\times n$, 其中，$m$ 为数据基数，$n$ 为时序维度; 其次，距离约束$\delta$对每个值来说过于宽泛，即该约束不能起到很好的过滤效果，如当 $n$ 很大时，往往$|x_i-y_i|<<\delta$, 最终导致了许多无效的数据扫描和计算.

6.2.2 曼哈顿距离

曼哈顿距离定义为 $D_m(X,Y)=|x_1-y_1|+|x_2-y_2|+\ldots+|x_n-y_n|$. 与欧式距离的处理流程相似，分块存储上令时间周期 $T_p=1$. 查询算法上，替换算法 1 中的距离函数并预估一个$\delta$值，因$|x_i-y_i|\leqslant D_m(X,Y)\leqslant\delta$, 故其余步骤无须改变. 与欧式距离的讨论同理，曼哈顿距离的处理方式仍然存在数据条数激增和距离约束宽泛等问题.

6.2.3 其他距离

其他距离如动态时间规整(dynamic time warping, DTW)、编辑距离(edit distance on real sequence, EDR)、带惩罚的编辑距离(edit distance on real penalty, ERP)、最长公共子串(longest common sub-sequence, LCSS)和弗雷歇距离(Fréchet distance)等，也可以很好地衡量两序列之间的相似程度，但这些距离的计算均涉及序列的全局信息，其计算公式均基于动态规划的思想，这样的特性使其无法适用于“分而治之”的处理思想，即无法将序列切割为子序列再聚合计算，无法很好地支持并行计算. 此外，即使将每条时序整体存为数据库的一行,因其距离计算并不涉及极值，且距离下界难以确定，故无法很好地、有针对性地设计其行键进行有效过滤，从而退化成了线性遍历的查询方法.

综上，这些距离函数并不适用于本文提出的基于分块存储的键值查询框架.

## 7 结论与展望

时序相似性查询是时序数据库的基本算子之一，其上层应用普遍对其有较高的时效性要求. 本文主要研究了基于键值数据库的时序数据存储和相似性查询问题，面向切比雪夫距离的相似性度量方式提出了一种基于分块表示的时序存储方法及其相应的分布式查询算法，同时提出了两种高效的剪枝策略，即极值剪枝和分块剪枝，实现了可提前过滤无效数据加快查询进程的算法 KV-Search，有效地解决了大数据背景下时序基数大、维度高和不断增长的三大挑战. 经实验对比, KV-Search 在效率和扩展性方面均优于基准实验.

注意到预估的$\delta$值大小对查询算法的影响较大，所以在今后的工作中，会重点研究如何更加精确地预估$\delta$值; 此外，本文仅使用切比雪夫距离作为衡量时序相似性的度量方式，具有一定局限性，如何基于键值数据库支持更多的相似性衡量指标并进行针对性优化，也是未来亟待解决的问题.

**References**:


[1] Zheng Y, Capra L, Wolfson O, Yang H. Urban computing: Concepts, methodologies, and applications. ACM TIST, 2014, 5(3): 1–55.

[2] Yuan JD, Wang ZH, Sun YG, Zhang W. *K*-nearest neighbor classifier for complex time series. Ruan Jian Xue Bao/Journal of Software, 2017, 28(11): 3002−3017 (in Chinese with English abstract). [doi: 10.13328/j.cnki.jos.005331]

[3] Feng YC, Jiang T, Li GH, Zhu H. Underlying techniques of efficient similarity search on time series. Chinese Journal of Computers, 2009, 32(11): 2107−2122 (in Chinese with English abstract).

[4] Ma RQ, Song BY, Ding LL, Wang JL. Dynamic matrix clustering method for time series events. Journal of Frontiers of Computer Science and Technology, 2020, 15(3): 468−477 (in Chinese with English abstract).

[5] Wang MX, Ding XO, Wang HZ, Li JZ. Correlation-based method for tracing multi-dimensional time series data anomalies. Journal of Frontiers of Computer Science and Technology, 2020, 15(11): 2142−2150 (in Chinese with English abstract).

[6] Ding XO, Yu SJ, Wang MX, Wang HZ, Gao H, Yang DH. Anomaly detection on industrial time series based on correlation analysis. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 726−747 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5907.htm [doi: 10.13328/j.cnki.jos.005907]

[7] Li R, He H, Wang R, Huang Y, Liu J, Ruan S, He T, Bao J, Zheng Y. Just: JD urban spatio-temporal data engine. In: Proc. of the IEEE 36th Int'l Conf. on Data Engineering. IEEE, 2020. 1558−1569.

[8] Li R, He H, Wang R, Ruan S, Sui Y, Bao J, Zheng Y. Trajmesa: A distributed NoSQL storage engine for big trajectory data. In: Proc. of the IEEE 36th Int'l Conf. on Data Engineering. IEEE, 2020. 2002−2005.

[9] Li R, He H, Wang R, Ruan S, He T, Bao J, Zhang J, Hong L, Zheng Y. TrajMesa: A distributed NoSQL-based trajectory data management system. IEEE Trans. on Knowledge and Data Engineering, 2021. https://ieeexplore.ieee.org/abstract/document/9430714

[10] Yi XW, Zheng Y, Zhang JB, Li TR. ST-MVL: Filling missing values in geo-sensory time series data. In: Proc. of the 15th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2016). AAAI, 2016. 2704−2710.

[11] Cha SH. Comprehensive survey on distance/similarity measures between probability density functions. Int'l Journal of Mathematical Models and Methods in Applied Sciences, 2007, 1(4): 300−307.

[12] Baskar SS, Arockiam L. C-LAS relief—An improved feature selection technique in data mining. Int'l Journal of Computer Applications, 2013, 83(13): 33−36.

[13] Yagoubi DE, Akbarinia R, Masseglia F, Palanas T. Massively distributed time series indexing and querying. IEEE Trans. on Knowledge and Data Engineering, 2018, 32(1): 108−120.

[14] Yang K, Ding X, Zhang Y, Chen L, Zheng B, Gao Y. Distributed similarity queries in metric spaces. Data Science and Engineering, 2019, 4(2): 93−108.

[15] Yang ZX, Tang N, Tang Y, Pan MM, Li DD, Ye XP. Temporal index and query based on timing partition. Ruan Jian Xue Bao/Journal of Software, 2020, 31(11): 3519−3539 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5826.htm [doi: 10.13328/j.cnki.jos.005826]

[16] Morse MD, Patel JM. An efficient and accurate method for evaluating time series similarity. In: Proc. of the SIGMOD Conf. ACM, 2007. 569−580.

[17] Wu WCH, Yeh MY, Pei J. Random error reduction in similarity search on time series: A statistical approach. In: Proc. of the ICDE. 2012. 858−869.

[18] Jin S, Keogh E. iSAX: Indexing and mining terabyte sized time series. In: Proc. of the 14th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2008. 623−631.

[19] Zoumpatianos K, Idreos S, Palpanas T. Indexing for interactive exploration of big data series. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2014. 1555−1566.

[20] Peng B, Fatourou P, Paris PT. The next destination for fast data series indexing and query answering. In: Proc. of the 2018 IEEE Int'l Conf. on Big Data (Big Data). 2018. 791−800.

[21] Michele L, Palpanas T. Scalable, variable-length similarity search in data series: The ULISSE approach. Proc. of the VLDB Endowment, 2018, 11(13): 2236−2248.

[22] Huijse P, Estevez PA, Protopapapas P, Principe J, Zegers P. Computational intelligence challenges and applications on large-scale astronomical time series databases. IEEE Computational Intelligence Magazine, 2014, 9(3): 27−39.

[23] Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing. In: Proc. of the 25th Int'l Conf. on Very Large Data Bases (VLDB). Morgan Kaufmann Publishers Inc., 1999. 518−529.

[24] Alghamdi N, Zhang L, Zhang H, Rundensteiner E, Eltabakh M. ChainLink: Indexing big time series data for long subsequence matching. In: Proc. of the 2020 IEEE ICDE. IEEE, 2020. 529−540.

[25] Ur RMH, Liew CS, Abbas A, Jayaraman P, Wah T, Khan S. Big data reduction methods: A survey. Data Science and Engineering, 2016, 1(4): 265−284.

[26] Popivanov I, Miller RJ. Similarity search over time-series data using wavelets. In: Proc. of the 18th Int'l Conf. on Data Engineering. IEEE, 2002. 212−221.

[27] Peng JL, Wang HZ, Li JZ, Gao H. Set-based similarity search for time series. In: Proc. of the SIGMOD Conf. ACM, 2016. 2039−2052.

[28] Vora MN. Hadoop-HBase for large-scale data. In: Proc. of the 2011 ICCS. IEEE, 2011. 601−605.

[29] Borthakur D. HDFS architecture guide. 2008. https://www.researchgate.net/publication/242787758_HDFS_architecture_guide

**附中文参考文献:**

[2] 原继东, 王志海, 孙艳歌, 张伟. 面向复杂时间序列的 *k* 近邻分类器. 软件学报, 2017, 28(11): 3002−3017. http://www.jos.org.cn/1000-9825/5331.htm [doi: 10.13328/j.cnki.jos.005331]

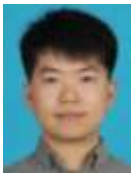
[3] 冯玉才, 蒋涛, 李国徽, 朱虹. 高效时序相似搜索技术. 计算机学报, 2009, 32(11): 2107−2122.

[4] 马瑞强, 宋宝燕, 丁琳琳, 王俊陆. 面向时间序列事件的动态矩阵聚类方法. 计算机科学与探索, 2020, 15(3): 468−477.

[5] 王沐贤, 丁小欧, 王宏志, 李建中. 基于相关性的多维时序数据异常溯源方法. 计算机科学与探索, 2020, 15(11): 2142−2150.

[6] 丁小欧, 于晟健, 王沐贤, 王宏志, 高宏, 杨东华. 基于相关性分析的工业时序数据异常检测. 软件学报, 2020, 31(3): 726−747. http://www.jos.org.cn/1000-9825/5907.htm [doi: 10.13328/j.cnki.jos.005907]

[15] 杨佐希, 汤娜, 汤庸, 潘明明, 李丁丁, 叶小平. 基于时序分区的时态索引与查询. 软件学报, 2020, 31(11): 3519−3539. http://www.jos.org.cn/1000-9825/5826.htm [doi: 10.13328/j.cnki.jos.005826]



**俞自生**(1996－), 男, 硕士生, CCF 学生会员, 主要研究领域为城市计算, 时空数据管理与分析, 分布式数据库.

**李瑞远**(1990－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为时空数据管理与挖掘, 城市计算.

**郭阳**(1997－), 女, 硕士, CCF 学生会员, 主要研究领域为机器学习, 计算机视觉, 数据挖掘.

**蒋忠元**(1988－), 男, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为复杂网络系统安全, 隐私保护, 社交网络,数据挖掘.

**鲍捷**(1985－), 男, 博士, CCF 专业会员, 主要研究领域为城市计算, 时空数据管理与挖掘.

**郑宇**(1979－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为时空数据挖掘, 城市计算.