# 云平台下时间序列数据并行化排列熵特征提取方法

杨 鹏[1,2],申洪涛[1,2],陶 鹏[2],冯 波[2],张洋瑞[2],王立斌[2]
(1. 国网河北能源技术服务有限公司,河北 石家庄 050000;
2. 国网河北省电力有限公司电力科学研究院,河北 石家庄 050000)

**摘要**:随着高级量测体系和各类监控系统的大规模建设发展,时间序列数据的规模呈指数级增长,在智能电网大数据中占有较大的比重。时间序列数据的特征提取是影响数据挖掘质量的关键步骤,在大数据背景下,传统的特征提取算法已无法满足海量数据处理的需求。结合云计算平台和 MaxCompute 大数据处理技术,设计实现了时间序列数据的表存储方法和并行化的时间序列数据排列熵特征提取算法。在云计算平台上采用不同规模的数据集对并行化算法进行测试,验证了并行化排列熵算法的正确性和高性能。

**关键词**:时间序列数据;排列熵;特征提取;并行算法;大数据;云计算
**中图分类号**:TM 732 **文献标识码**:A DOI:10.16081/j.issn.1006-6047.2019.04.032

## 0 引言

随着智能电网建设的不断深入,发电、输变电、配电、用电以及调度各个生产过程累积了大量的数据,充分挖掘智能电网大数据的价值具有重要的意义[1]。智能电网大数据的分析包括数据收集、数据清洗、特征提取、数据挖掘等一系列过程,其中特征提取是影响数据挖掘质量的关键步骤。尤其对于时间序列数据而言,通过特征提取可以将数据从高维空间转变到低维空间,达到降维和提升分类性能的目的。在大数据背景下,数据分析的各个环节都可以成为影响数据分析整体性能的瓶颈。特征提取过程以原始采集数据作为输入,并将特征量作为输出至数据挖掘过程,因此相较于数据挖掘过程,特征提取过程最容易成为影响整体性能的瓶颈,迫切需要借助大数据技术实现高性能并行化的特征提取算法。

智能电网大数据具有规模大、类型多样的特点,包括各类结构化、非结构化和半结构化数据。由于支撑智能电网的基础是对组成电网的各个环节和部件进行实时监测,因此积累了大量的时间序列数据,并在智能电网大数据中占有非常大的比重。时间序列数据本身具有高维度特性,因此不易直接用于数据挖掘,大多需要先进行有效的特征提取。

排列熵[2]算法作为一种较新的衡量时间序列数据复杂程度的方法,受到了国内外相关学者的高度关注。目前,排列熵已经在设备故障诊断[3]、信号异常检测[4]、变压器局部放电特征提取[5]、脑电图信号特征提取[6]、生物基因序列识别[7]、气候系统复杂性度量[8]、语音监测[9]、纹理图像分析[10]等众多领域得到了广泛的应用,并且效果明显,具有较好的通用性。排列熵在电力系统中也得到了一定的应用:文献[11]应用排列熵进行电力系统故障信号分析;文献[12]将小波与排列熵相结合,用于行波故障测距研究;文献[13]利用排列熵和变分模态分解进行输电线路的故障诊断。但在智能电网环境下,面对海量的智能电网大数据,在单机环境下应用的传统排列熵算法的计算性能已经无法满足海量数据处理的需求,其在存储容量以及系统扩展性方面也面临技术挑战。

目前,对时间序列数据的存储和处理方面的研究主要采用 Hadoop 开源大数据技术。文献[14]研究了基于 HBase 的电力设备高速采样数据的存储方法;文献[15]研究了基于 Hadoop 的输变电设备状态监测大数据存储优化与并行处理方法。这些方法大多基于自建的 Hadoop 平台,存在集群规模小、维护困难等问题。公有云具有免维护、弹性伸缩和按需租用等优点,并在互联网、生物、医学、电子政务等领域得到广泛的应用。

大数据计算服务(MaxCompute)是阿里云提供的 PByte/EByte 级数据存储和处理平台,其提供了海量的结构化和非结构化数据存储、并行编程框架 MapReduce、图计算模型 Graph 及数据进出通道 Tunnel 等数据处理组件,非常适用于海量历史数据的存储和批量并行数据分析。在数据存储方面,MaxCompute 的逻辑层采用表(Table)和分区(Partition)的方式组织数据,便于使用结构化查询语言 SQL(Structured Query Language)接口进行数据访问;在物理层则基于分布式文件系统实现了数据的分布式三副本存储,具有极高的数据可靠性和可用性;在访问接口方面,对 SQL、用户自定义函数 UDF(User Defined Function)、MapReduce 和 Graph 均具有很好的支持,目前已经在大型互联网企业的数据仓库和商务智能 BI(Business Intelligence)分析、电子商务网站的交易分析、用户特征和兴趣挖掘等领域得到了大规模的应用。目前 MaxCompute 在电力系统中的应用报道较少,文献[16]结合 MaxCompute 研究了

**收稿日期**:2018-04-23;**修回日期**:2019-01-16

变压器局部放电数据的相位分析方法。

本文结合 MaxCompute 大数据技术,设计实现了时间序列数据的 MaxCompute 表存储方法;基于 MaxCompute 提供的并行编程框架 MapReduce,设计实现了并行化的排列熵算法,实现了海量时间序列历史数据的并行化排列熵特征提取。

## 1 时间序列数据处理总体流程

时间序列数据分析的过程主要包括明确数据分析目的、数据收集、数据加工、数据分析和分析结果展示等环节。数据分析环节通常是指使用 SQL/命令方式进行数据的统计分析或者使用机器学习方法进行数据挖掘;分析结果展示则是通过报表或者数据可视化等手段展示数据分析的结果。时间序列数据分析的整体流程如图 1 所示。本文的主要工作是数据收集环节之后的格式转换、数据存储以及特征提取,这为之后的数据分析奠定了基础。

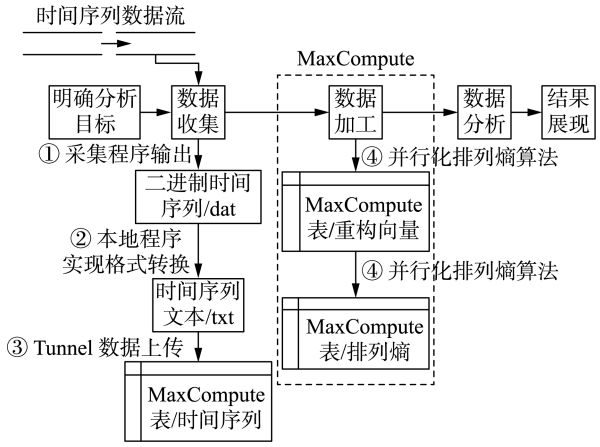


**图 1 时间序列数据分析整体流程**

Fig.1 Overall process of time series data analysis

图 1 中,原始的时间序列数据流被收集之后,通常会以二进制文件方式存储于采集设备本地文件系统。为了将数据上传至 MaxCompute,需要将二进制格式转换文本格式,如 txt 文档。同时,文本格式要与上传前设计好的 MaxCompute 表结构相匹配,匹配格式如图 2 所示。

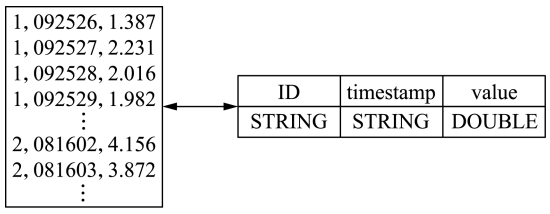


**图 2 文本文件和表的格式匹配**

Fig.2 Format matching between text file and table

可以使用 SQL DDL 语句完成创建表的操作,之后使用 Tunnel 工具进行数据上传。基于并行编程框架 MapReduce 设计实现了并行化的排列熵算法。算法在本地 Eclipse 开发环境调试正确后,需要导出 jar 包,并以资源(Resource)的方式上传至 MaxCompute,并使用 jar 命令执行。算法的输入数据均来自于 MaxCompute 表,排列熵的计算结果也输出至 MaxCompute 表中。上层应用系统可以使用 MaxCompute 提供的 SQL 接口直接读取表中的结果数据,用于后续的计算或者展示。

## 2 时间序列数据的 MaxCompute 表存储设计

时间序列数据主要是通过监测系统获取的一系列时间点上的观测值,时间单位可以大至 h 或 d,小至 ms 或 μs,因此经年累积的时间序列可以有很大的规模。传统的时间序列存储方法包括文本文件存储或者关系型数据库、实时数据库的存储方式。文本文件存储是较为通用的方式,在大数据环境下,目前主要是采用分布式文件系统实现大文本的存储,典型的系统包括 Hadoop 分布式文件系统 HDFS(Hadoop Distributed File System)等。关系型数据库则存在存储容量小和扩展性差的问题;实时数据库则更倾向于监测数据的写入和实时处理性能,并通过数据压缩算法解决存储容量问题。MaxCompute 是阿里云提供的数据仓库产品,在用户层使用二维表的方式存储数据,在底层则基于分布式文件系统以分布式文件方式存储。这种方式为用户提供了简单的接口,支持简单的 SQL 访问和并行编程,有效屏蔽了分布式存储细节,同时又实现了大数据的高效存储。因此,基于 MaxCompute 表的存储方式与文件存储、关系型数据库存储均具有很大的不同。

MaxCompute 存储数据的基本单位是表,因此在上传并存储数据之前需要先进行表模式的设计。文本形式的波形信号数据一般组织为按行存储,即每行存储 1 条用于排列熵计算的时间序列数据。根据数据来源和计算目的的不同,各条时间序列的长度不等。当使用文本存储时,通常 1 条数据占 1 行,对 1 条数据的长度没有限制。但 MaxCompute 表的宽度被限制为小于 1 024 列,因此对于单条数据长度大于 1 024 个数据点的情况,则不能按行存储。本文使用按列存储模式存储时间序列数据。排列熵计算过程中产生的重构向量使用按行存储的模式,如图 3 所示。首先需要将行存储的文件进行格式化,形成按列存储的文件。列存储文件需要与 MaxCompute 上的原始时间序列数据表的列进行匹配,才能执行 Tunnel 数据上传操作,如表 1 所示。表中的 ID 字段代表不同的时间序列信号标识,表示不同的排列熵计算使用的数据。

对时间序列数据执行排列熵计算的步骤 1 是向量重构,重构后的结果数据存储在 MaxCompute 的重构向量表中。重构向量的维数 $m$ 通常小于 1 024,因

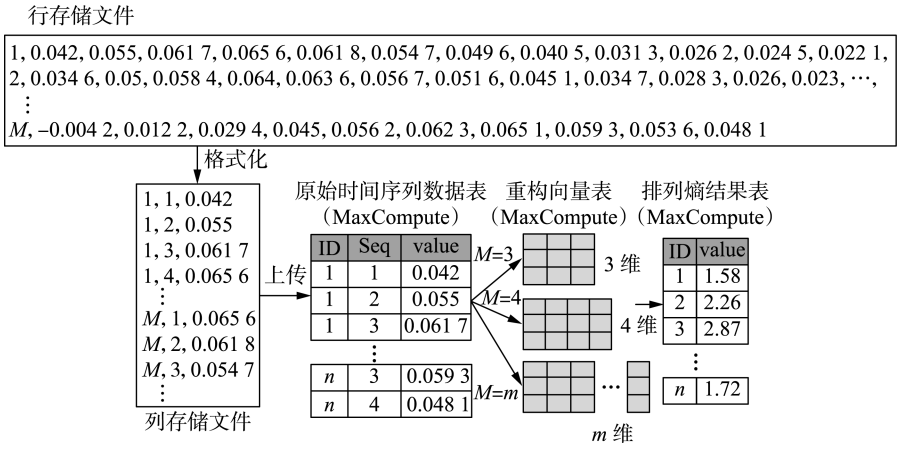


**图 3　时间序列数据的存储模式**

Fig.3　Storage mode of time series data

**表 1　数据上传操作**

Table 1　Data upload operation

| 命令 | 操作 | 功能 |
|---|---|---|
| Tunnel | Tunnel upload d:\data\sequence.txt seq_table | 将 d:\data\sequence.txt 上传至表 seq_table |

此可以进行有效的数据存储。根据计算维度需求的不同,需要分别创建多张重构向量表。以 3 维存储表为例,创建存储表的 SQL DDL 描述如表 2 所示。

**表 2　3 维重构向量表的 DDL 描述**

Table 2　DDL description of 3-dimensional vector table

| 命令 | DDL | 功能 |
|---|---|---|
| Create Table | Create Table if not exist d3vector( ID String,d1 double, d2 double,d3 double )life cycle $n$; | 创建 3 维向量表 |

在表 2 中,创建了包含 4 个字段的表 d3vector,其中第一个字段 ID 表示时间序列数据的标识,用于区分不同的排列熵计算;d1—d3 标识向量的 3 个维度值。其他维度表的定义模式与表 2 相同。用于存储排列熵计算结果的表模式固定,只包括波形信号标识 ID 和排列熵的值 2 个维度。

排列熵的计算结果也存储在 MaxCompute 表中,包含 ID 和排列熵结果这 2 列。根据后续数据分析的需要,可以使用数据集成服务将排列熵计算结果导出至其他存储系统,包括关系型数据库、非关系型数据库(NoSQL)、对象存储等,或者直接参与在 MaxCompute 上的后续数据分析任务。

## 3　基于 MapReduce 的并行化排列熵算法

### 3.1　排列熵

排列熵算法是一种度量时间序列复杂性的方法,算法描述如下。

设 1 维时间序列 $\boldsymbol{X}$ 如式(1)所示。

$$\boldsymbol{X}=[x(1),x(2),\cdots,x(n)] \tag{1}$$

计算 1 维时间序列 $\boldsymbol{X}$ 的排列熵,需要的输入数据包括 $\boldsymbol{X}$、重构维数 $M$ 以及延迟时间 $\tau$。计算过程包括相空间重构、相空间矩阵编码统计、排列熵计算 3 个步骤。

**a.** 相空间重构。

基于相空间重构延迟坐标法对 $\boldsymbol{X}$ 中的任一元素 $x(i)$ 进行相空间重构,对每个采样点取其连续的 $M$ 个样点,得到点 $x(i)$ 的 $M$ 维空间重构向量 $\boldsymbol{X}_i$,如式(2)所示。

$$\boldsymbol{X}_i=[x(i),x(i+1),\cdots,x(i+M-1)\tau] \tag{2}$$

其中,$M$ 为重构维数;$\tau$ 为时间延迟。则序列 $\boldsymbol{X}$ 的相空间矩阵 $\boldsymbol{M}_{\text{at}}$ 可以用式(3)表达。

$$\boldsymbol{M}_{\text{at}}=\begin{bmatrix} x(1) & \cdots & x(M) \\ \vdots & & \vdots \\ x(n-M\tau+\tau) & \cdots & x(n) \end{bmatrix} \tag{3}$$

**b.** 相空间矩阵编码统计。

式(3)中的每一行就是 $x(i)$ 的重构向量 $\boldsymbol{X}_i$,对重构向量中的各元素按照升序进行排列,得到升序序列 $\boldsymbol{X}_i'$ 如式(4)所示。

$$\boldsymbol{X}_i'=\begin{bmatrix} x(i+(j_1-1)\tau) \\ x(i+(j_2-1)\tau) \\ \vdots \\ x(i+(j_M-1)\tau) \end{bmatrix}^{\mathrm{T}} \tag{4}$$

$$x(i+(j_1-1)\tau)\leqslant x(i+(j_2-1)\tau)\leqslant\cdots\leqslant x(i+(j_M-1)\tau)$$

对向量进行排序后,将所得向量元素的位置序列记为 $\{j_1,j_2,\cdots,j_M\}$,然后将该序列作为重构向量的编码,得到向量空间编码矩阵 $\boldsymbol{M}_{\text{at}}'$。$\{j_1,j_2,\cdots,j_M\}$ 是全排列 $M!$ 中的一种,可以对序列 $\boldsymbol{X}$ 各种排列情况的出现次数进行统计。

**c.** 排列熵计算。

根据步骤 **b** 的统计结果,将各种排列情况出现的相对频率作为其概率,记为 $p_1$、$p_2$、$\cdots$、$p_k$($k\leqslant M!$),利用式(5)计算序列的排列熵 $P_{\text{e}}$。

$$P_e = -\sum_{i=1}^{k} p_i \log_2 p_i \tag{5}$$

排列熵算法的流程如图 4 所示。

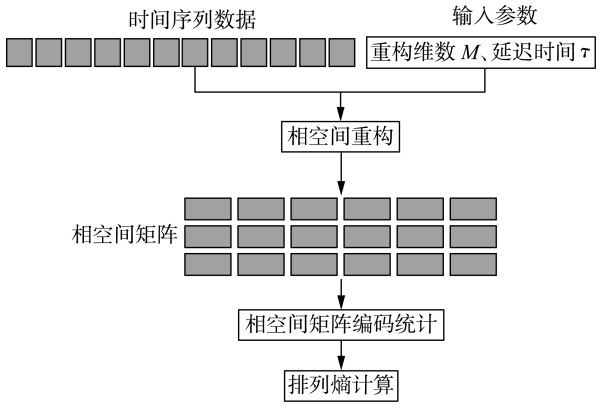


**图 4 排列熵算法流程**

Fig.4 Flowchart of permutation entropy algorithm

### 3.2 并行化排列熵算法的实现

利用阿里云 MaxCompute 提供的并行计算框架 MapReduce 设计实现了并行化排列熵算法，用于海量时间序列数据的排列熵计算。计算过程包含纵表转横表和排列熵计算这 2 个阶段。

首先需根据原始的时间序列数据纵表，计算得到重构向量表(横表)。该计算过程使用自定义聚合函数和 SQL 执行。3 维重构向量表的 SQL 描述如表 3 所示。表中，使用了自定义聚合函数 collect( ) 完成重构向量元素的选择，并使用 ID 进行分组计算。计算结果插入重构向量表 d3vector。

**表 3 3 维重构向量表的 SQL 描述**

Table 3 SQL description of 3-dimension reconstructed vector table

| 命令 | DDL | 功能 |
|---|---|---|
| Insert into | Insert into d3vector<br>Select ID,<br>collect( value,$i$,$\tau$),<br>collect( value,$i$+1,$\tau$),<br>collect( value,$i$+2,$\tau$)<br>from sequence_table group by ID; | 向 3 维向量表插入数据 |

在获得重构向量后，利用 MapReduce 完成后续的排列熵计算，整个并行算法处理数据的过程包含 Map、Combine、Reduce1、Reduce2 这 4 个任务，它们之间的连接关系如图 5 所示。

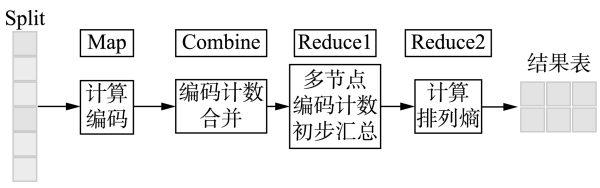


**图 5 并行化排列熵算法任务连接关系**

Fig.5 Connectivity of tasks in parallel permutation entropy algorithm

Map 任务以分布式并行的方式计算编码，并在 Combine 任务中完成编码计数合并，以减少网络通信开销。然后，不同节点的 Combine 任务的执行结果通过网络发送至 Reduce1 任务进行初步的统计汇总。由于不同的排列编码统计结果在 Reduce1 中被多个节点分别汇总，所以还需要一个进一步的汇总任务 Reduce2。Reduce2 中接收到了所有的排列编码统计计数，然后才能完成排列熵的计算。Map、Combine 和 Reduce 的处理逻辑包含了排列熵算法的全部实现。处理流程可描述如下。

**a.** 输入数据的预处理。输入数据(重构向量表)被框架 MapReduce 自动拆分成多个数据分片(Split)，每个 Split 是大小相等的数据块，每个 Split 作为单个 Map 任务的输入被处理，同时有多个 Map 任务并行，并且按照相同的程序逻辑执行对 Split 的处理。

**b.** Map 任务。在 Map 任务中，对输入的相空间重构向量进行编码计算。对重构向量中的各元素按照升序进行排列，得到式(4)所示的升序序列 $\boldsymbol{X}'_i$。对向量进行排序后，将所得向量元素的位置序列记为$\{j_1, j_2, \cdots, j_M\}$，将该序列作为重构向量的编码。根据框架 MapReduce 约定，Map 任务的输出必须是<key,value>格式。本程序中，key = $\{j_1, j_2, \cdots, j_M\}$，value = 1，表示这个编码出现了 1 次，以便于在后续的过程中进行累加计数。key 值决定了一条输出数据将会被发送的 Reduce 任务。key 值和 Reduce 任务是多对一的关系，具有相同 key 值的数据会被发送给同一个 Reduce 任务，单个 Reduce 任务可以接收多个 key 值的数据。Map 任务的处理逻辑如表 4 所示。

**表 4 Map 任务的处理逻辑**

Table 4 Process logic of task Map

| 接口名称 | Map |
|---|---|
| Input | $M$ 维重构向量表<br><RecordNum,{vID,(相空间重构向量)}> |
| 逻辑功能 | 对相空间重构向量进行编码计算 |
| Output | <{vID,(相空间重构向量)},record(1)> |

**c.** Combine 任务。Combine 任务用于对 Map 任务所在计算节点的本地数据进行合并操作，即将相同的向量编码进行合并，合并后的 key 值仍然是向量编码，value 的值等于相同向量编码出现的次数之和。Combine 任务由框架调用，将具有相同 key 值的数据进行聚合，Combine 任务的输入、输出参数必须与 Reduce 任务保持一致。Combine 任务的处理逻辑如表 5 所示。

**d.** Reduce1 任务。在利用 Combine 执行完成本地的合并操作后，框架将负责将多个计算节点的中间计算结果根据 key 值对数据进行派发。Reduce1

表 5　Combine 任务的处理逻辑

Table 5　Process logic of task Combine

| 接口名称 | Combine |
|---|---|
| Input | <{vID,(相空间重构向量)},<br>Iterator<Record> value> |
| 逻辑功能 | 对 Iterator<Record> value 中的次数进行累加,<br>累加结果记为 add_value |
| Output | <{vID,(向量编码)},add_value> |

任务会将相同向量编码数据发送到同一个 Reduce1 任务。在 Reduce1 任务中完成向量编码的统计计数,其输出的 key 取值为 1,目的是将累加结果全部都送至同一个 Reduce2 进行最后的汇总工作。输出的 value 值为该向量编码的累加次数。Reduce1 任务的处理逻辑如表 6 所示。

表 6　Reduce1 任务的处理逻辑

Table 6　Process logic of task Reduce1

| 接口名称 | Reduce1 |
|---|---|
| Input | <{vID,(向量编码)},<br>Iterator<Record> add_value> |
| 逻辑功能 | 对 Iterator<Record> value 中的次数进行累加,<br>累加结果记为 result_value |
| Output | <1,Record{{vID,(向量编码)},result_value}> |

**e.** Reduce2 任务。由于 Reduce1 任务不能获取所有向量编码及其累加值,因此还需要一个额外的 Reduce2 任务,其接收所有 Reduce1 任务的输出,并且进行汇总计算排列熵,同时进行归一化处理。Reduce2 产生的计算结果将输出至图 3 所示的排列熵结果表。Reduce2 任务的处理逻辑如表 7 所示。

表 7　Reduce2 任务的处理逻辑

Table 7　Process logic of task Reduce2

| 接口名称 | Reduce2 |
|---|---|
| Input | <1,Iterator<Record> {{vID,(向量编码)},<br>result_value}> |
| 逻辑功能 | 提取 Iterator<Record> {{vID,(向量编码)},<br>result_value}中向量编码累积值,<br>根据式(5)计算排列熵 |
| Output | Record{vID,result_value} |

## 4　实验分析

### 4.1　实验环境

基于阿里云 MaxCompute 进行本文实验,综合使用 MaxCompute 生态系统的多种组件。使用 MaxCompute SQL DDL 执行数据表创建,使用 Tunnel 工具执行数据上传;使用 Dataworks 完成并行排列熵程序的资源上传,使用 SQL DML 执行纵表转横表的重构向量计算,使用 Dataworks 的 ODPS_MR 节点任务执行并行排列熵计算。组件的层次关系如图 6 所示。图中,MaxCompute 作为基础的存储和计算引擎,Dataworks 作为上层数据开发环境。各组件功能的详尽描述请参见阿里云官网帮助文档的描述。

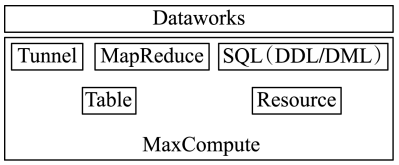


图 6　MaxCompute 组件的层次关系

Fig.6　Hierarchical relation of MaxCompute components

### 4.2　计算性能分析

在实验室采集了三相绕组变压器在正常状态下的振动数据作为实验数据。振动数据的采样频率为 10 kHz,为了验证并行化排列熵算法的性能,利用不同大小的数据集进行多次实验,数据集描述如表 8 所示。

表 8　数据集

Table 8　Data sets

| 数据集 ID | 记录数/条 | 文件存储容量/MByte | MaxCompute 表存储容量/MByte |
|---|---|---|---|
| D1 | 100 | 8.9 | 2.7 |
| D2 | 200 | 17.9 | 5.6 |
| D3 | 500 | 44.8 | 13.2 |
| D4 | 1 000 | 89.6 | 27.3 |
| D5 | 2 000 | 179.2 | 56.2 |
| D6 | 5 000 | 448.2 | 135.8 |
| D7 | 10 000 | 896.5 | 259.1 |
| D8 | 20 000 | 1 793.3 | 562.2 |
| D9 | 50 000 | 4 482.5 | 1 280.2 |
| D10 | 100 000 | 8 965.6 | 2 716.1 |

表 8 所示的数据集中,每条振动信号数据包含 10 240 个采样值(1 个采样周期的时间序列数据),采样值为浮点数。

分别在单机环境下和 MaxCompute 平台上完成变压器振动信号数据排列熵的计算,并与利用 logview 测量算法的执行时间、CPU 资源占用、内存占用、实例数量(instance count)进行对比。单机算法的执行环节为 Inteli3-2350M CPU,4.00 GB 内存。测试数据从 100 条递增至 100 000 条,测试数据集的最大规模为 8 GByte。

首先对单机算法和并行算法的执行时间进行了对比,结果如图 7 所示。

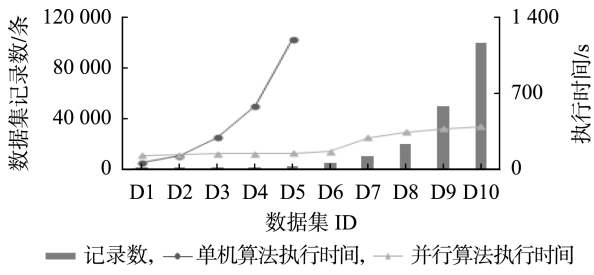


图 7　执行时间对比

Fig.7　Comparison of execution time

从图 7 中可看出,单机环境下算法的执行时间与数据规模基本呈正比关系增长,当数据集记录

达到1 600条时,执行时间已超过20 min,执行缓慢,已经不再适用于处理更大规模的数据集;并行算法的执行时间则非常平稳,随着数据集记录的变化增长非常缓慢,可见其对海量数据的处理能力强。

并行算法的执行时间非常平稳的主要原因是计算资源的动态调整。并行算法计算过程中CPU资源和内存资源的变化趋势如图8所示。

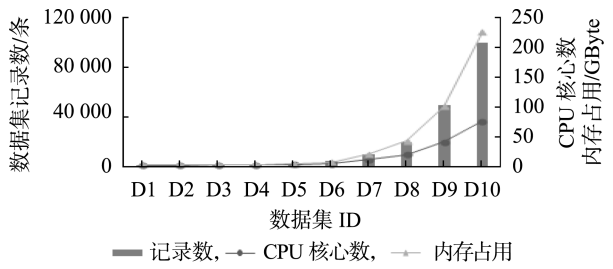


**图8 并行算法的CPU核心数和内存占用**

Fig.8 CPU cores and memory usage of parallel algorithm

从图8中可看出,CPU和内存资源的分配数量随着分析数据规模的增长而增长,计算资源的弹性调整保证了数据处理执行时间的平稳。平台分配的计算资源与数据规模呈正比增长,前期数据规模较小,未达到计算瓶颈,计算资源数量变化平缓;从处理D7开始,计算资源数量增长速度大于线性增长。在计算层面,实例数量和计算资源也进行了相应的调整,随着所处理数据规模的增长而增长。实例数量的变化趋势如图9所示。

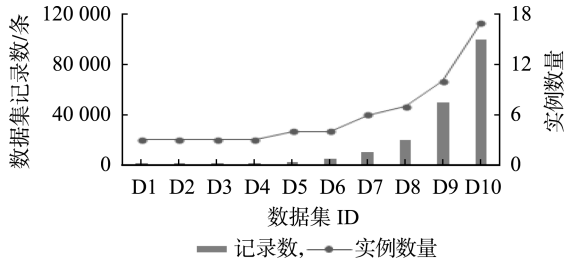


**图9 实例数量的变化趋势**

Fig.9 Change of instances count

## 5 结论

在智能电网建设中,时间序列数据具有体量大、维度高的特点,需要大数据技术对其进行有效的存储和高效的特征提取,才能更好地用于后续的数据分析。本文研究了基于云平台的适用于时间序列数据的存储方法和基于SQL/MapReduce的并行数据处理方法,设计实现了并行化排列熵特征提取算法,基于变压器振动信号数据,在MaxCompute上验证了所提算法的性能,给出了分析结果;同时给出了阿里云MaxCompute生态系统各组件的配合使用方法。该数据处理过程可以有效地扩展至其他各类智能电网大数据的处理。

## 参考文献:


[1] 宋亚奇,周国亮,朱永利. 智能电网大数据处理技术现状与挑战[J]. 电网技术,2013,37(4):927-935.
SONG Yaqi,ZHOU Guoliang,ZHU Yongli. Present status and challenges of big data processing in smart grid[J]. Power System Technology,2013,37(4):927-935.

[2] BANDT C,POMPE B. Permutation entropy:a natural complexity measure for time series[J]. Physical Review Letters,2002,88(17):174102.

[3] 刘永斌,龙潜,冯志华,等. 一种非平稳、非线性振动信号检测方法的研究[J]. 振动与冲击,2007,26(12):131-134.
LIU Yongbin,LONG Qian,FENG Zhihua,et al. Detection method for nonlinear and non-stationary signals[J]. Journal of Vibration and Shock,2007,26(12):131-134.

[4] 冯辅周,饶国强,司爱威,等. 排列熵算法研究及其在振动信号突变检测中的应用[J]. 振动工程学报,2012,25(2):221-224.
FENG Fuzhou,RAO Guoqiang,SI Aiwei,et al. Research and application of the arithmetic of PE in testing the sudden change of vibration signal[J]. Journal of Vibration Engineering,2012,25(2):221-224.

[5] 张蒙,朱永利,张宁,等. 基于变分模态分解和多尺度排列熵的变压器局部放电信号特征提取[J]. 华北电力大学学报(自然科学版),2016,43(6):31-37.
ZHANG Meng,ZHU Yongli,ZHANG Ning,et al. Feature extraction of transformer partial discharge signals based on varitional mode decomposition and multi-scale permutation entropy[J]. Journal of North China Electric Power University(Natural Science Edition),2016,43(6):31-37.

[6] NICOLAOU N,GEORGIOU J. Detection of epileptic electroencephalogram based on permutation entropy and support vector machines[J]. Expert Systems with Applications,2012,39(1):202-209.

[7] SUN X L,ZOU Y,NIKIFOROVA V,et al. The complexity of gene expression dynamics revealed by permutation entropy[J]. BMC Bioinformatics,2010,11(1):607-621.

[8] 郝成元,吴绍洪,李双成. 排列熵应用于气候复杂性度量[J]. 地理研究,2007,26(1):46-52.
HAO Chengyuan,WU Shaohong,LI Shuangcheng. Measurement of climate complexity using permutation entropy[J]. Geographical Research,2007,26(1):46-52.

[9] LUO Y J,YE X M,FAN Q G. Texture image analysis based on discrete Fourier transform and permutation entropy[J]. Electronic Science and Technology,2011,24(1):9-11.

[10] 吴秀良,范影乐,钱诚,等. 基于排列组合熵的语音端点检测技术研究[J]. 计算机工程与应用,2008,44(1):240-242.
WU Xiuliang,FAN Yingle,QIAN Cheng,et al. Application of permutation entropy measure in detecting speech[J]. Computer Engineering and Applications,2008,44(1):240-242.

[11] 李从善,刘天琪,李兴源,等. 基于排列熵算法的电力系统故障信号分析[J]. 电子科技大学学报,2015,44(2):233-238.
LI Congshan,LIU Tianqi,LI Xingyuan,et al. Power system fault signal analysis based on permutation entropy algorithm[J]. Journal of University of Electronic Science and Technology of China,2015,44(2):233-238.

[12] 何军娜. 基于小波及排列熵奇异性检测的行波故障测距研究[D]. 南昌:华东交通大学,2015.
HE Junna. Research on transmission line traveling wave fault location based on wavelet transform and the permutation entropy[D]. Nanchang:East China Jiaotong University,2015.

[13] 姜媛媛,刘朋,王康,等. 基于变分模态分解和排列熵的输电线路故障诊断[J]. 电子测量与仪器学报,2017,31(7):1025-1030.
JIANG Yuanyuan,LIU Peng,WANG Kang,et al. Fault diagnosis of transmission lines based on variational mode decomposition and permutation entropy[J]. Journal of Electronic Measurement and Instrumentation,2017,31(7):1025-1030.

[14] 宋亚奇,刘树仁,朱永利,等. 电力设备状态高速采样数据的云存储技术研究[J]. 电力自动化设备,2013,33(10):150-156.
SONG Yaqi,LIU Shuren,ZHU Yongli,et al. Cloud storage of power equipment state data sampled with high speed[J]. Electric Power Automation Equipment,2013,33(10):150-156.

[15] 宋亚奇,周国亮,朱永利,等. 云平台下输变电设备状态监测大数据存储优化与并行处理[J]. 中国电机工程学报,2015,35(2):255-267.
SONG Yaqi,ZHOU Guoliang,ZHU Yongli,et al. Storage optimi-zation and parallel processing of condition monitoring big data of transmission and transforming equipment based on cloud platform[J]. Proceedings of the CSEE,2015,35(2):255-267.

[16] 朱永利,李莉,宋亚奇,等. ODPS 平台下的电力设备监测大数据存储与并行处理方法[J]. 电工技术学报,2017,32(9):199-210.
ZHU Yongli,LI Li,SONG Yaqi,et al. Storage and parallel processing of big data of power equipment condition monitoring on ODPS platform[J]. Transactions of China Electrotechnical Society,2017,32(9):199-210.

**作者简介:**


杨　鹏

杨　鹏(1975—),男,石家庄辛集人,高级工程师,主要研究方向为电力计量和电力系统自动化;

申洪涛(1969—),男,河北邯郸人,高级工程师,主要研究方向为电力计量;

陶　鹏(1979—),男,河南焦作人,高级工程师,硕士,主要研究方向为电力计量和大数据分析(**E-mail**:hbdyytp@163.com)。

## Parallel permutation entropy feature extraction method for time series data based on cloud platform

YANG Peng[1,2],SHEN Hongtao[1,2],TAO Peng[2],FENG Bo[2],ZHANG Yangrui[2],WANG Libin[2]

(1. State Grid Hebei Energy Technology Service Limited Company,Shijiazhuang 050000,China;
2. State Grid Hebei Electric Power Research Institute,Shijiazhuang 050000,China)

**Abstract**:With the large-scale construction and development of AMI(Advanced Metering Infrastructure) and various monitoring systems,the size of time series data grows exponentially,which occupies a large proportion in the smart grid big data. The feature extraction of time series data is a key step that affects the quality of data mining. Traditional feature extraction algorithms can no longer meet the requirements of mass data processing in the context of big data. The table storage method and feature extraction algorithm based on parallel permutation entropy are designed and implemented for time series data by combining with the cloud computing platform and MaxCompute big data processing technology. Different scale data sets are tested on the cloud computing platform,and results verify the accuracy and high performance of the parallel permutation entropy algorithm.

**Key words**:time series data;permutation entropy;feature extraction;parallel algorithm;big data;cloud computing