大规模时间序列分析框架的研究与实现

滕 飞[1),2)] 黄齐川[1)] 李天瑞[1)] 王 晨[3)] 田春华[3)]

[1)](西南交通大学信息科学与技术学院 成都 610031)

[2)](中铁一院轨道交通工程信息化国家重点实验室 西安 710043)

[3)](清华大学大数据系统软件国家工程实验室 北京 100084)

摘 要 工业互联时代,每天数以亿计的传感器源源不断生成时间序列,用以记录工业设备的温度、振动、压力、曲度和张力等参数.如何从这些非结构化的时间序列中挖掘出有价值信息,并运用于状态监测、故障诊断和控制决策,引起了广泛的关注和研究.随着数据规模日益增长,能够提供较为完备数据分析算法库的主流单机环境如Matlab、R 等已无法较好地应对大规模时间序列分析场景下的数据处理需求.而现有的并行分析算法数量有限,常与平台相互绑定,更换平台需对算法进行二次开发,可扩展性较差.本文旨在设计一种通用的近似解分析框架,支持第三方算法快速实现并行化,解决因数据规模过大而导致的算法适用性问题.分析框架主要包含任务划分、治理和合并三个步骤.任务划分通过冗余保留了数据的局部相关性,生成相互独立的子任务,减少分布式节点之间的数据通信和同步开销.对于任务划分问题,本文提出了近似解代价模型,得到了最优的任务划分方案.基于 Spark 平台设计并实现了原型系统,实验结果表明,该系统在确保分析结果准确性的前提下,其加速能力随着并行程度保持近似线性的增长,解决了单机算法的数据规模受限问题.同时,该系统易于集成与扩展,使数据分析人员免于算法重复开发.

关键词 时间序列;算法并行化;近似解;分治;Spark

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2020.01279

An Analysis Framework for Large-Scale Time Series

TENG Fei[1),2)] HUANG Qi-Chuan[1)] LI Tian-Rui[1)] WANG Chen[3)] TIAN Chun-Hua[3)]

[1)](*School of Information Science and Technology, Southwest Jiaotong University, Chengdu* 610031)

[2)](*State Key Laboratory of Rail Transit Engineering Informatization, China Railway First Survey and Design Institute Group, Xi'an* 710043)

[3)](*National Engineering Laboratory for Big Data Software, Tsinghua University, Beijing* 100084)

Abstract In the era of Industrial Internet, an explosive volume of time series is continuously generated by various sensors, such as temperature, vibration, pressure, inclination and strain. It is crucial to analyze these unstructured time series to extract valuable information for state monitoring, fault diagnosis and control decision. In specialized fields, abundant data mining and analysis tools have been provided to seek the potentially available knowledge from time series. For example, R and Matlab contain a great quantity of algorithms for matrix manipulation. However, these single-machine environments are no longer effective, or even invalid, especially when the volume and velocity of time series is extremely big. Although there are already several commercials or open-source software for distributed computing, most of them only provide a limited number of parallel algorithms, and these parallel algorithms are platform-dependent. The same algorithm should be developed repeatedly on different platforms, that is to say, it is difficult to extend one

收稿日期:2018-09-04;在线发布日期:2019-07-10.本课题得到国家重点研发计划项目(2018YFB1701502)、四川省科技计划(2019YJ0214)资助.**滕 飞**,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为并行计算、云计算等.E-mail:fteng@swjtu.edu.cn.**黄齐川**,硕士,主要研究方向为云计算、互联网技术等.**李天瑞**,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为数据挖掘与知识发现、云计算与大数据、粒计算与粗糙集等.**王 晨**,博士,研究员,主要研究领域为工业大数据、数据平台技术等.**田春华**,博士,研究员,主要研究领域为数据分析与运筹优化.

parallel algorithm from its original platform to other distributed platforms. This paper studied a Large-scale Time Series Analysis Framework (LTSAF) as a general-purpose tool to parallelize third-party algorithms in a quick manner, so that the existing algorithms are able to analyze the massive time series data efficiently. Based on the insight of divide and conquer, LTSAF proposed an approximate solution if the exact solution cannot be obtained within a feasible period. The solution includes three steps, division, data-parallel computation and combination. The analysis task is firstly divided into a number of subtasks, and each subtask contains a segment of original time series and necessary redundancy to keep data locality. Division makes the subtasks data independent, so the synchronization overhead of intermediate results is greatly reduced. The independent subtasks are small enough, which are then solved by stand-alone algorithms directly. The solutions of subtasks are finally combined to create an approximate solution to the original problem by removing the dirty parts. From theoretical aspects, a time-space cost optimization model is established to determine how to divide the subtasks efficiently, and the optimal length of segmentation is deduced to make a balance between time and space efficiency. This paper also developed a prototype Spark system that could transplant the verified algorithms. Experiments showed the approximate solution made stand-alone algorithms applicable to deal with large-scale time series while obtaining precise results. The parallel algorithms greatly decrease the processing time of large datasets and the scaleup of these algorithms are no longer limited by size of input data. In addition, because subtasks are completely data-independent of one another, the speedup of parallel algorithms increased approximately linearly with the degree of parallelism, in that case, it is easy to estimate the overall computation time of massive time series datasets in advance. As a cross-langrage and cross-platform approach, this prototype system is facile to integrate third-party libraries, so that system users can avoid repetitive development of the existing algorithms, but focus on data analysis.

Keywords time series; data parallel; approximate solution; divide and conquer; spark

1 引 言

随着物联网、大数据等新一代信息技术逐渐向工业领域渗透,现代工业设备中安装了数以亿计的传感器,来探测温度、压力、振动和噪声.传感器产生的数据经过解码和转换后形成一维或高维的时间序列,其体量远大于企业中计算机和人工产生的数据.与网络大数据相比,时间序列数据分析对于准确性和时效性要求更高.一方面,时间序列分析已形成了丰富的机理、模型与算法,需要加以充分利用.另一方面,由于数据规模巨大,很多串行化分析算法受限于机器算力常在实际应用场景中失效,无法做到实时分析、诊断和预测.

现有的分析工具可大致分为两类:一类用于小规模的复杂分析,另一类用于大规模的简单统计分析.前者的典型代表为 SPSS[1]、Matlab[2] 和 R 等,包含了大量的时间序列分析算法和开源库,但这些算法适用于传统的单机分析.后者的代表为 Hadoop[3]、Spark[4] 等分布式计算平台,可以存储和计算大规模的非结构化数据,但用于时间序列分析的算法不够丰富,无法开展一些比较复杂的分析工作[5].目前有不少学者致力于研究分布式的矩阵运算工具,且多选择 MapReduce 或 Spark 作为执行环境对串行的分析算法进行二次开发[6],如 Mahout[7]、System ML[8]、MADlib[9]、MLlib[10] 等.这些算法库将适用于小规模数据的串行分析方法部分扩展应用到了更大规模的数据上,数量比较有限.面对多变的应用问题需求,通常需要重复编写和测试新的并行化算法,给数据分析和算法设计工作带来很大的不便和困难.

由此可见,大规模时间序列分析系统不仅要具备复杂数据分析能力还需具有高效的数据处理能力,解决数据规模受限问题.此外,在并行化时间序列分析算法时,应充分利用成熟的矩阵运算工具箱和算法库,而非全部重新设计开发.

本文设计了一种通用的大规模时间序列分析框架(Large-scale Time Series Analysis Framework，LTSAF)，重点研究了算法并行化中任务划分问题，建立了近似解代价模型，推导出最优任务划分序列长度.该框架支持第三方算法快速实现并行化，求取满足应用需求的近似解.同时，采用 Spark 计算引擎开发了原型系统，验证了分析框架可以大幅降低时间序列分析的时间开销，解决算法适用的数据规模受限的问题.

本文第 2 节介绍相关工作；第 3 节构建任务划分的理论模型和大规模时间序列分析框架设计原理；第 4 节以 Spark 为例介绍该框架的一种原型系统实现方案；在第 5 节中，介绍实验环境和实验数据，同时较为详细地讨论并行参数实验、可处理数据规模实验和正确性验证实验的结论.

2　相关工作

时间序列通常按照既定频率生成，写入过程持续时间长，具有数据平稳的特点.以风速传感器为例，其采样频率为 1 Hz，一年累积的时间序列长度将达到 3155 万.其特点有以下两点：

一是具有全局关联性弱等特点.由于时间序列采样频率较高，通常具有较强的周期性，全局关联性较弱，适合分段进行存储.例如风速数据通常是以月、季或年为周期，特别久远的数据对近期趋势的影响较小.

二是具有局部性相关性强等特点.时间序列中单点数据价值有限，无法体现出数据的变化规律和趋势，因而需将一定范围内的多个数据点作为分析对象.此外，连续数据点除包含固有数据趋势及周期等物理属性外，还因场景的差异性携带大量业务属性，因此，多维度切入与分析将有利于对时间序列数据的全面认知.

时间序列分析的常用工具有 SPSS、Matlab、SAS[11]、Minitab[12]等，商业产品的特点是界面友好且对于数据生命周期提供较为完整的服务，但由于商业性质的排他性和封闭性，此类工具的兼容性和扩展性较差.目前被工业界和学术界广泛使用的开源工具箱为 R 和 Python，两者皆提供了标准的时间序列处理算法.由于支持开放的编程环境，研究人员可将最新的研究成果和优化算法编制函数供他人使用，第三方算法库的更新速度比一般商业软件更快.随着数据规模的增长，单台高性能计算机已经无法支持复杂度较高的算法，需要采用分布式计算模型，以提高计算效率和扩大数据处理规模.

基于分布式构架的数据处理平台近年来发展迅猛，典型的商业平台有亚马逊 Machine Learning[13]，微软 DMTK，IBM System ML，阿里云 DTPai①，腾讯 Angel②，百度 BML③等.分布式平台根据分析任务不同可大致分为三类.第一类基于 Hadoop 和 Spark 平台，针对特定问题设计专用的并行算法，如序列模式挖掘[14]、序列查找[15]、序列预测[16]，这些算法的设计重点在于合并并行任务的结果，而较少关注任务划分.除了专用算法，也有比较通用的算法库，如 MLlib、Mahout 等，这些算法库算法数量依然比较有限，而且对于经典算法不能有效利用，必须进行重新开发，可扩展性较差.第二类基于 BSP(Bulk Synchronous Parallel)模式的并行图计算框架，如 Pregel[17]、GraphX[18]、GraphLab[19]等，此类模型处理的数据对象为图，与时间序列呈现局部相关不同，图数据自身结构具有不规则特点，并行计算时进行任务划分很容易造成负载不均匀，并导致极为庞大的通讯量.图划分也因此受到了越来越多的研究者的关注，提出了基于聚类[20]、矩阵分解[21]、分层[22]的方法等.这些方法的复杂度较高，不适于作为分布式计算中的时间序列划分方法.冗余是图划分的一种有效方法[23]，通过复制特定节点和连边到其他计算机，用空间换时间，来减少网络通信的代价.第三类采用参数服务器架构的学习系统[24]，如 CMU Parameter Server④、Petuum 以及深度学习 Tensorflow On Spark[25]等，此类系统主要针对机器学习任务，与第一类系统的不同之处在于采用了延迟同步并行计算模型，旨在解决单一的主节点难以存储大规模参数集问题，缓解了数据规模增长造成的节点通信瓶颈[26].

综上可知，目前国内外针对时间序列分析通用框架的研究较少，更多的是面向机器学习、图计算和深度学习应用的分布式计算平台，以上平台在进行并行计算时，重点优化数据通信和参数传递问题，较少关注任务划分.受限于分布式框架，并行算法往往不重视数据划分工作，一般按照经验或框架默认值将数据集直接分成固定大小的数据分块，缺少对于数据分块之间的相互关联性的考虑.这种简单划分的方

① Aliyun DTPai. https://data.aliyun.com/
② Tencent Angel. https://github.com/Tencent/angel
③ Baidu ML. http://cloud.baidu.com/product/bml.html
④ CMU. http://parameterserver.org/

式导致分布式计算框架合并任务的过程变得复杂且开销很大,不能实现完全的数据并行.此外,无论是商业平台还是开源平台,并行算法与平台相互绑定,算法很难互用,也无法有效利用第三方算法库,用户更换平台学习成本较高.RHadoop[27]和 SparkR[28]虽然一定程度上允许调用 R 算法包,但由于不能完成任务划分,无法发挥出分布式平台的并行计算能力.

在机器算力的约束下,大规模时间序列分析的精确解往往难以获得.考虑到时间序列的自身特点,如周期性明显、局部相关性等,本文提出求取近似解来换取更高的效率,通过任务划分实现完全的数据并行,降低数据规约的通信开销.该方法对分布式计算平台不做限定,具有较好的通用性.

3 基于数据并行的时间序列分析模型

3.1 任务划分问题

分治是处理大规模数据的通用思路,一般包括划分、治理和合并三个步骤.关于并行算法的研究大都针对特定问题设计治理和合并的方法,力求降低合并时通信的开销.本文重点研究了划分方法,为简化治理、合并方法和实现数据并行奠定了基础.

数据并行的关键是任务划分.为避免分布式节点之间频繁的数据通信和同步,将时间序列切分成多个独立的子序列,使各子序列间不存在计算依赖关系,实现数据并行.任务划分需重点关注以下两点.一方面,时间序列的访存模式呈现局部相关特点,在分治处理数据时需要增加冗余保留不同分段序列之间的关联性.另一方面,分治处理虽能解决复杂度过高导致的性能瓶颈,但过细粒度的分段将极大损失原有时间序列所包含的特征属性,例如周期性等信息.

任务划分采用的分段冗余方案如图 1 所示.将单机无法处理的整体数据划分为若干个小片段,片段之间通过冗余保持局部相关,并具有一定的相互独立性.冗余是对相邻原始时间序列的重复,通过保留与相邻分段的联系,降低了分段对数据产生的干扰.以分段 3 为例,左冗余 3 和右冗余 3 均取值于原始时间序列.

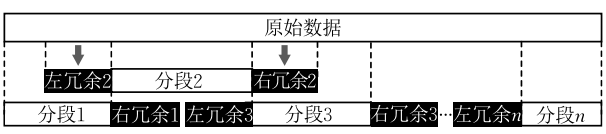


图 1 任务划分方案

任务划分的核心问题是确定合适的分段长度 S 和冗余范围 R.在分段长度足够大时,冗余范围越大,数据关联性保持越好.此外,冗余 R 与时间复杂度和空间复杂度正相关,需要找出满足正确性要求的 R 最小值.当 R 固定,S 越小,冗余信息占据存储空间越大,而 S 越大,计算时间越长.

本文提出代价理论模型,将分段长度 S 的求解问题转化为时空代价的优化问题.原始序列先划分为相互独立的子序列,再添加冗余信息生成并行任务.假设原始序列长度为 m,子序列长度为 S,并行任务长度为 $n=S+2R$,共计 m/S 个并行任务.

时间代价刻画的是在采用相同算力的条件下,经分段冗余处理之后的数据计算时间与未经处理的数据计算时间的比值.采用时间复杂度表示计算时间,时间代价为

$$Cost_T(S)=\frac{\frac{m}{S}(T(n)+C)}{T(m)+C} \tag{1}$$

$T(n)$是单个并行任务的计算时间;$T(m)$是原始任务的计算时间;C 为除计算耗时外的其他时间开销.时间序列分析算法的复杂度比较高,通常大于线性阶 $O(n)$,故 S 越大,时间代价越高.时间代价的取值范围是 $0<Cost_T(S)<1$.

空间代价表示存储经分段冗余处理之后的数据与存储未经处理的原始数据的空间之比.空间代价为

$$Cost_H(S)=\frac{1}{2}\left(\frac{n}{S}-1\right) \tag{2}$$

n/S 代表并行任务由于添加冗余所增加的空间存储比例.由于有效计算长度 S 应大于冗余信息 R,可知 $1<n/S<3$.将空间代价进行归一化,得到式(2),S 越大,空间代价越小.空间代价的取值范围是 $0<Cost_H(S)<1$.

为达到时空效率的平衡,建立多目标优化问题:

$$Cost(S)=\omega Cost_T(S)+(1-\omega)Cost_H(S) \tag{3}$$

ω 为时空权重,取值范围是 $0<\omega<1$,代价函数的优化目标可表示为

$$\begin{aligned}&\min\ Cost(S)=\omega Cost_T(S)+(1-\omega)Cost_H(S)\\&\text{s.t.}\ \ m\gg n\geq S\geq 2R\end{aligned} \tag{4}$$

3.2 最优任务划分方案

代价函数的自变量包括分段长度和时空权重,其目标极值可通过求解分段长度 S_{best} 实现.本文以复杂度 $O(m^2)$ 的时间序列分析算法为例展示参数求解过程.首先对式(4)求偏导.

当 $\partial Cost(S)/\partial\omega=0$ 时,

$$S_{\text{best}}=\sqrt{\frac{R(m^2+C)}{m}}-C-2R \tag{5}$$

当 $\partial Cost(S)/\partial S=0$ 时，

$$\omega_{\text{best}}=\frac{m^2R+RC}{m^2R+RC+mS^2-4mR^2-mC} \tag{6}$$

易得 $(S_{\text{best}},\omega_{\text{best}})$ 为驻点，可使 $Cost(S)$ 取得最小值. 海量数据场景下的时间序列长度 m 的取值很大，远远大于分段长度 S，其计算时间将远远超过计算工具启动时间，故有

$$S=\sqrt{mR}-2R \tag{7}$$

$$\omega=\frac{m^2R}{m^2R+mS^2-4mR^2} \tag{8}$$

考虑到 $m\gg R$，式(7)进一步简化得

$$S=\sqrt{mR} \tag{9}$$

此外，令 $\theta=1/\omega$，对式(8)求偏导得

$$\frac{\partial\theta}{\partial m}=-\frac{1}{m^2}\cdot\frac{S^2-4R^2}{R}<0 \tag{10}$$

易证明 ω 与 m 呈递增关系，m 越大，ω 也越大，$m\to\infty$ 时，$\lim\omega=1$. 由此可知，随着数据规模扩大，对时间效率的要求也不断提高.

3.3　通用框架

基于上述模型，本文提出一种通用的大规模时间序列分析框架 LTSAF，包括划分、治理和规约三个阶段，如图 2 所示.

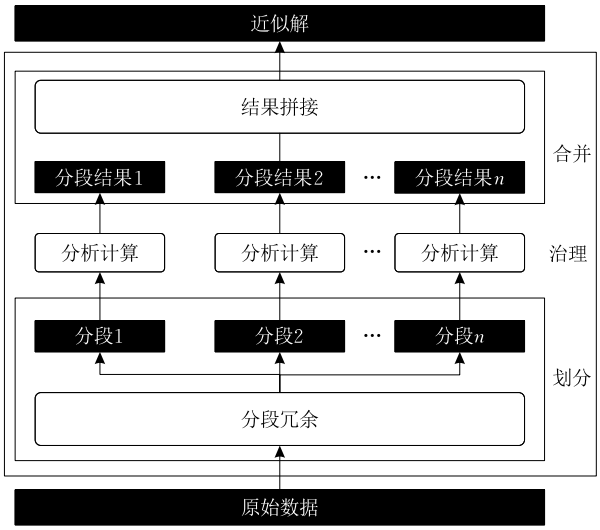


图 2　大规模时间序列分析框架

任务划分生成相互独立的子任务. 治理阶段对数据并行的子任务进行分析计算，分析算法既可调用第三方算法，也可根据情况开发. 规约阶段合并子任务的解，生成最终结果. 相比于原始序列直接进行分析计算所得的精确解，经过 LTSAF 框架处理所得的结果被称为近似解. 近似解之于精确解的相似程度衡量了 LTSAF 框架的正确性，本文采用如式(11)定义的皮尔逊相关系数来表示近似解与精确解的相似度.

$$\rho_{xy}=\frac{\sum_{n=0}^{\infty}x_ny_n}{\left(\sum_{n=0}^{\infty}x_n^2\sum_{n=0}^{\infty}y_n^2\right)^{\frac{1}{2}}} \tag{11}$$

其中，x_n 和 y_n 是两个时间序列，ρ_{xy} 在 0 到 1 之间取值，代表两者的相似程度，ρ_{xy} 越大，两个时间序列的近似程度越高.

4　原型系统设计

作为一种通用的大规模时间序列分析框架，LTSAF 可以在不同的计算引擎部署实现，本文基于 Spark API 为例实现了 LTSAF 原型系统，如图 3 所示. 该原型系统还可以接入 Spark 生态圈其他的组件和模块，快速实现并行算法的集成与扩展.

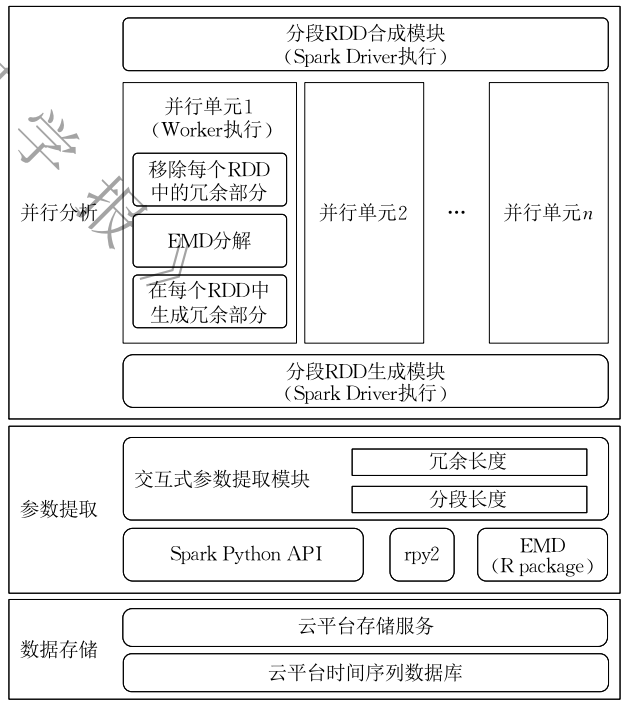


图 3　基于 Spark 平台的 LTSAF 原型系统

4.1　数据存储层

传感器监测数据是一种典型的时间序列，具有规模大、维度高等特点，通常需要对其进行深度分析才能获得有价值的信息. 本文将传感器产生的时间序列数据作为主要研究的数据对象.

数据存储层选用 HBase[29] 作为分布式数据库.

表 1 展示了 HBase 表模式. HBase 以列簇的形式存放数据,根据采样频率设置时间窗口,每个时间窗口内按照时间偏移量对采样点的数据进行存储,必要时还可对采样序列进行压缩.

表 1 传感器时间序列数据的表模式

时间序列 ID	时间窗口	传感器编号	采样通道	压缩值
30001	698752642	7541Z	1	{k:[0,3],v:[5.0,−3.0]}
30002	698752643	7541X	2	{k:[3,6],v:[−11.0,18.0]}
30002	698752644	7543Z	2	{k:[3,6],v:[−4.0,21.0]}
30003	698752645	7544X	3	{k:[0,3,6],v:[0.0,−2.0,6.0]}
30004	698752646	7545X	4	{k:[0,3,6,9],v:[−2.0,7.0,8.0,−5.0]}

4.2 参数提取层

参数提取层用于确定冗余范围和分段长度,这是决定任务划分性能优劣的关键. EMD(经验模态分解,Empirical Mode Decomposition)时间序列分析算法有效应用于各个工程领域,例如海洋状态监测、机械故障诊断、土木结构模态识别等. 下文以 EMD 举例详细介绍分析框架各个环节的具体实现方式.

LTSAF 参数提取流程如图 4 所示. 冗余范围 R 的寻优目标是查找满足预设阈值的最小 R 取值. 设定分段长度为远大于信号周期,逐步调整冗余范围的取值,生成不同的任务划分方案. 再经过并行治理和合并规约等步骤,对比近似解与精确解,选择满足相似度预设阈值的 R 最小值. 由于时空代价函数随着 R 增大而单调上升,也可以通过改变步长来加速寻优过程. 此外,最优分段长度 S 可通过优化时空代价的目标函数(4)取得解析解.

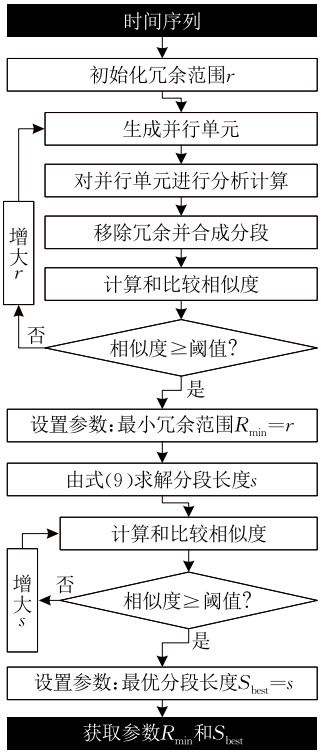


图 4 参数提取流程

4.3 并行分析层

并行分析流程如图 5 所示. 根据 Spark 集群的实际计算能力,结合获取的参数 S 与 R,划分为数据独立的子任务,并行单元执行各自的分析任务,删除冗余部分产生的不准确的结果,保留有效部分并按照顺序进行拼接,合并形成完整序列的分析结果.

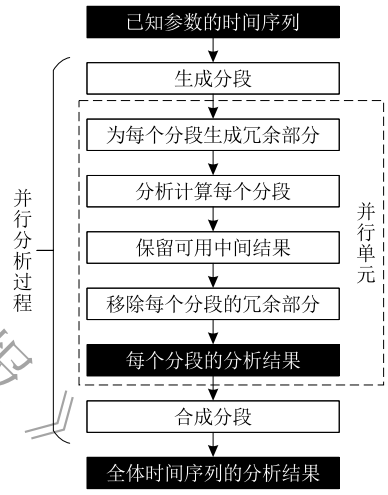


图 5 并行分析流程

对于 EMD 算法,如果任务划分不合理,会加剧端点效应,导致分析结果不准确. 如图 6 所示,通过冗余,可以将端点效应控制在一定范围内,移除污染的计算结果,抑制端点效应的扩散.

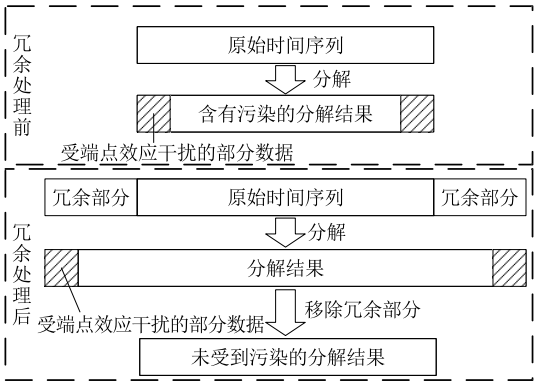


图 6 端点效应

4.4 算法扩展

LTSAF 原型系统通过 Spark Python API 和 rpy2 包[①]集成了 CRAN(Comprehensive R Archive Network)发布的 EMD 包[②]，现有 EMD 算法无需重新开发调试，便可利用 Spark 集群强大的计算能力，扩大数据处理规模，提高计算效率，实现路径如图 7 的深色区域所示.

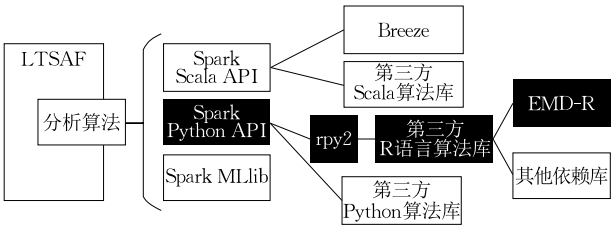


图 7　算法扩展

图 7 是 Spark 原型系统的算法集成和扩展方案. 基于 Scala 或 Python 语言所开发的第三方算法库(如 Scala 中的 Breeze、Python 中的 NumPy 和 SciPy 等)，都可直接集成于 LTSAF 原型系统. 同时，原型系统可通过 rpy2 编程接口来使用 R 语言的生态圈中提供的时间序列算法库，或者扩展机器学习算法库 MLlib.

5 实　验

本节在基于 Spark API 开发的 LTSAF 原型系统上开展实验，选取 EMD 和 SSA(奇异谱分析，Singular Spectrum Analysis)算法对时间序列数据进行分析，并对分析的正确性和可处理数据规模进行验证.

5.1 实验环境

单节点实验环境为四核心的 Intel Core i7 2630QM@2.00 GHz CPU、8 GB RAM 和 500 GB HDD，软件环境为 RStudio 1.1.463. Spark 集群中每个节点的硬件环境为四核心的 Intel Xeon E3-1230@3.20 GHz CPU、8 GB RAM 和 500 GB HDD，软件环境为 Hadoop 2.6.0 和 Spark 1.6.1，集群含有 8 个配置相同的节点.

5.2 数据集

仿真时间序列包括周期信号分量和趋势分量，如式(12)所示.

$$x=100\sin(50\pi t)+60\sin(150\pi t)+10\cos(30\pi t)+2t+n(t) \tag{12}$$

其中，$n(t)$是基于原始调制信号添加的高斯白噪声，信噪比为 40 dB. 仿真时间序列采样频率为 1000 Hz，取各组成成分周期的最大值作为时间序列周期，即 $T=0.067$ s，每个时间序列周期包含 67 个采样点.

此外，本文还采用高速列车走行部振动监测时间序列作为验证数据集，采样频率为 243 Hz.

5.3 并行参数实验

并行参数包括冗余范围及分段长度，本实验以 EMD 及 SSA 算法为例，详细介绍实验方法、过程和结果.

5.3.1 冗余范围

仿真时间序列的周期为 0.067 s，也即一个周期内所包含 67 个数据点. 固定分段长度为 $15T$，冗余范围取值如式(13)

$$R=0.25\times n\times T,\ n=1,2,\cdots,60 \tag{13}$$

将原始时间序列的计算结果设定为精确解，采用 LTSAF 得到的分解结果为近似解，比较精确解与 EMD 近似解的相似度，能量较高的第一分量和第二分量得相似度超过 99.99%，能量最低的第三分量在冗余 $3T$ 时，相似度可达 99.26%.

本文将最小分量的相似度 99%作为阈值. 该阈值通常与具体的应用场景有关. 一般来说，阈值与寻优开销成正比，也即阈值越高，寻优开销越大，且寻优边际成本增速加快，趋近于无穷大. 图 8 展现了 n 在[0,20]范围内的相似度值，如图所示，$R=3T$ 后继续提高 n 值，相似度增长速度越来越缓，无限逼近 1.

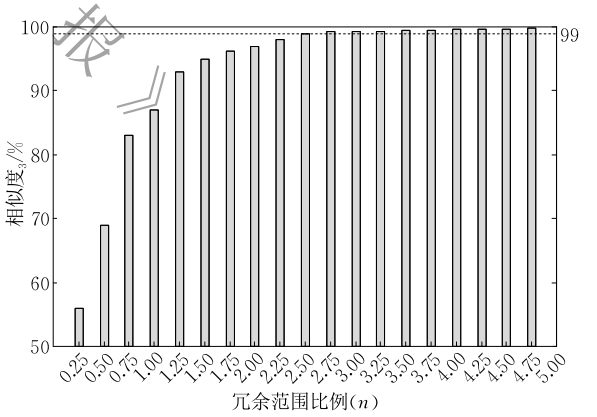


图 8　第三分量的相似度比较

对于 SSA 算法，冗余范围取值 $2T$ 即可确保三个分量的近似解与精确解的相似度在 99.99%以上.

5.3.2 分段长度

分段长度可对时空代价函数优化求解取得. EMD 算法在 CRAN 库中实现复杂度为 $O(n^2)$，代入式(7)可得

① rpy2-R in Python. https://rpy2.bitbucket.io/
② EMD: Empirical Mode Decomposition and Hilbert Spectral Analysis. https://cran.r-project.org/web/packages/EMD/

$$S=\sqrt{mR}-2R=\sqrt{1.005\times10^5\times201}-2\times201\approx4082 \tag{14}$$

图 9 展示了不同分段长度对时空代价的影响，空间代价随着分段长度增加快速下降，时间代价随着分段长度增加缓慢上升，而时空代价呈现凹函数特点，存在全局最小值，此时分段长度为 4082. 当分段长度超过最优取值后，时间代价比空间代价具有更高的重要性. 此参数条件下并行与直接分解结果相似度为 99.3%.

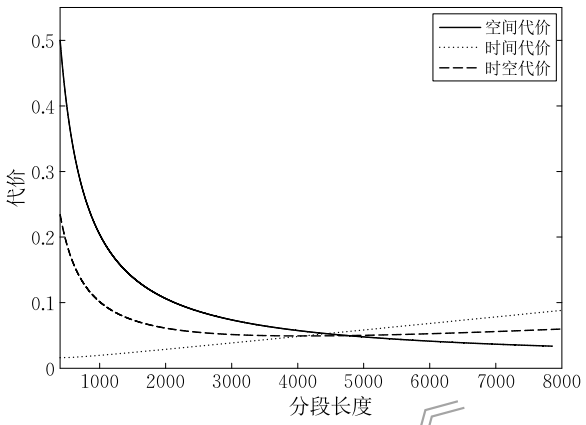


图 9 分段长度对时空代价的影响

5.4 数据规模实验

本实验使用 RStudio、SparkR 和 LTSAF 对仿真时间序列进行 EMD 和 SSA 分解. 消耗时间包括了任务从提交至完成的所有时间，OOM 表示内存溢出，TIMEOUT 表示超过 24 h. LTSAF 参数按照 5.3 小节的实验结果进行设置，分段长度 S 取值 4082，冗余范围 R 取值 201.

EMD 算法的实验结果如表 2 所示. 采用并行方式的 LTSAF 和 SparkR 的计算效率和处理数据规模显著优于 RStudio. 当数据集规模增加至 32 MB，且时间序列长度为 4 500 000 时，RStudio 已无法在 24 h 内输出有效结果；拥有 8 节点计算集群的 SparkR 耗时 113 s，而 LTSAF 仅用时 24 s. 将 RStudio 的分析结果视为精确解，LTSAF 和 SparkR 的近似分解结果分别与精确解相比，三个分量的相似度均为 99.3%，满足阈值要求.

表 2 EMD 算法在 LTSAF、SparkR 和 RStduio 上的计算时间

数据量/MB	EMD 基于 LTSAF 运行时间/s				EMD 基于 SparkR 运行时间/s				EMD 基于 RStudio 运行时间/s
	2 节点	4 节点	6 节点	8 节点	2 节点	4 节点	6 节点	8 节点	
8	**29**	**20**	**14**	**12**	125	70	59	42	1893
32	**78**	**51**	**31**	**24**	411	245	166	113	9071
128	**268**	**146**	**94**	**81**	1277	644	521	405	TIMEOUT
512	**1101**	**552**	**349**	**310**	4491	2515	1623	1238	TIMEOUT
1024	**2299**	**1123**	**654**	**545**	8211	4217	2769	2134	OOM

需要特别说明的是，通过 SparkR 调用 EMD-R 包，其性能表现与单节点 RStudio 相似，因为 SparkR 不能自动完成数据划分，实现并行计算. 按照并行框架修改之后，SparkR 的并行能力才得以体现，但运行效率依然低于 LTSAF.

LTSAF 系统中负载均衡的实现主要依赖于 Spark 平台自带的调度功能. 原始数据经分段冗余操作完成任务划分，独立的计算任务由 Spark 动态地平衡分配给各个节点，从而使得 LTSAF 系统有了负载均衡能力.

SSA 算法进一步分为两类. NUTRLAN 方法仅计算部分奇异值，其计算精度低但计算效率高. SVD 方法计算全部奇异值，其计算精度高但计算效率低[31]. LTSAF 和 SparkR 均采用 SVD 算法测试计算耗时，其结果如表 3 所示. 在数据集规模为 8 MB，RStudio 已无法在 24 h 内计算出有效结果. 虽然 NUTRLAN 算法效率略高，但同样无法处理 32 MB 以上的数据集. 基于分布式的 LTSAF 和 SparkR 有较好的运算能力，在数据集大小为 1024 MB 时，8 节点的 SparkR 需要 11 744 s 完成分析运算，而 LTSAF 耗时仅 5406 s. LTSAF 分解的近似解分别与 RStudio 的精确解相比，三个分量的相似度均在 99.99%以上.

表 3 SSA 算法在 LTSAF、SparkR 和 RStduio 上的计算时间

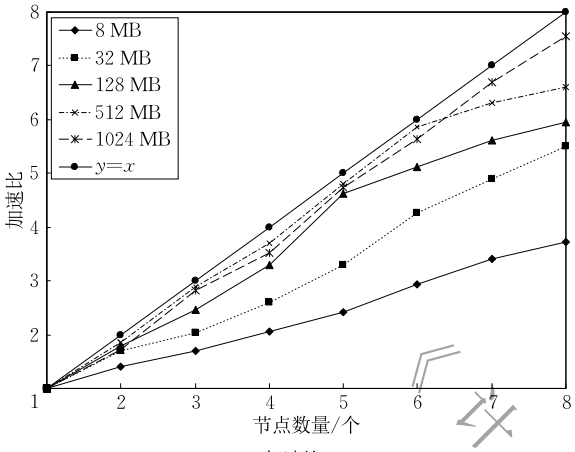
数据量/MB	SVD 基于 LTSAF 运行时间/s				SVD 基于 SparkR 运行时间/s				SVD 基于 RStudio 运行时间/s	NUTRLAN 基于 RStudio 运行时间/s
	2 节点	4 节点	6 节点	8 节点	2 节点	4 节点	6 节点	8 节点		
8	**243**	**129**	**77**	**62**	333	188	107	75	OOM	257
32	**837**	**485**	**282**	**220**	1863	876	656	493	OOM	413
128	**3070**	**1483**	**1064**	**789**	7858	4704	2705	2109	OOM	OOM
512	**13223**	**5686**	**4293**	**3038**	29171	14987	8034	6807	OOM	OOM
1024	**29427**	**11567**	**8175**	**5406**	52070	20983	15209	11744	OOM	OOM

5.5　并行性能实验

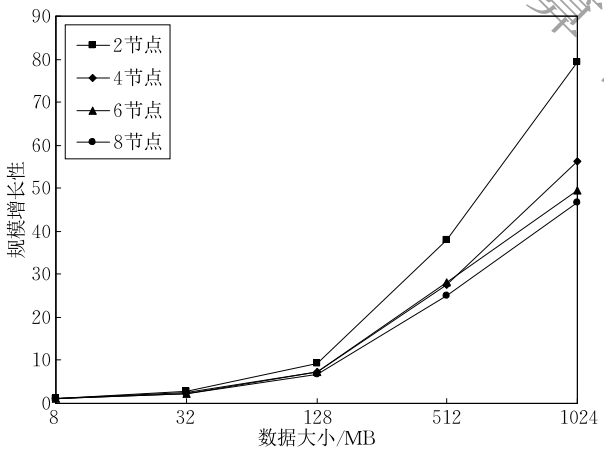
并行性能主要考察比较了加速比、规模增长性和可扩展性三个经典指标.

5.5.1　加速比

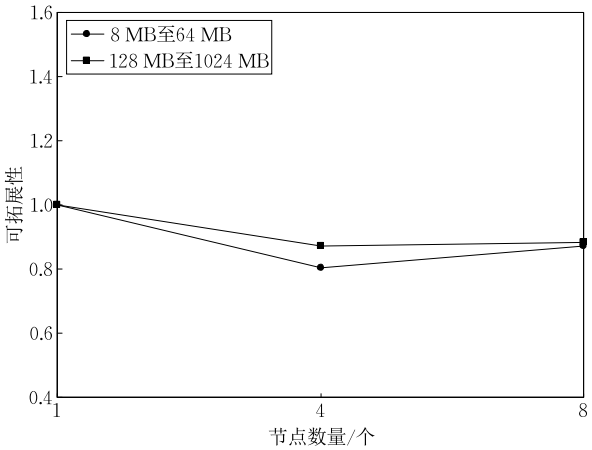
EMD 算法和 SSA 算法的加速比 Speedup 测试结果如图 10(a)～图 13(a)所示.对于较大规模数据集(如 1024 M),二者的加速比接近于理想状态($y=x$),当节点数目增加至 8 时,加速比均为 7.5 上下浮动.



(a) 加速比

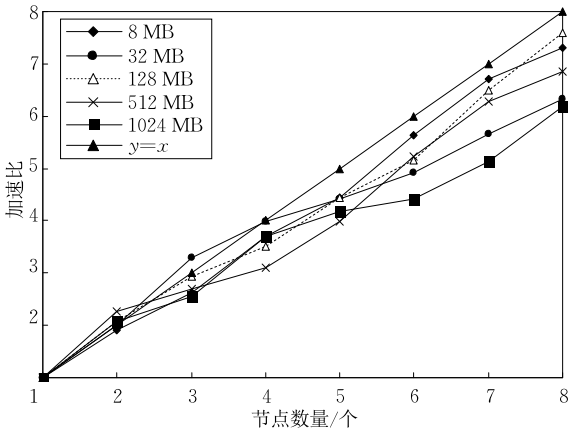


(b) 规模增长性

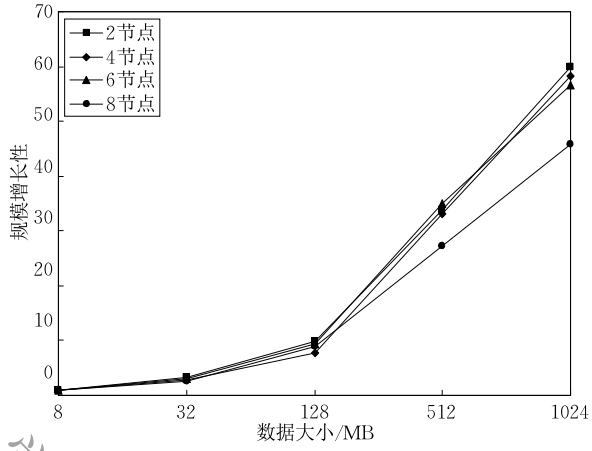


(c) 可扩展性

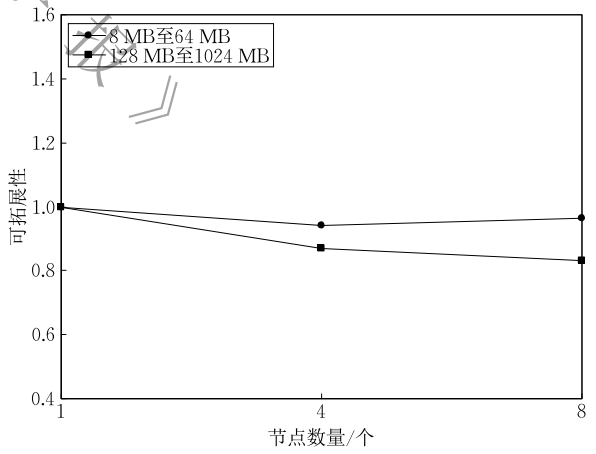
图 10　LTSAF 运行 EMD 算法的并行指标



(a) 加速比



(b) 规模增长性

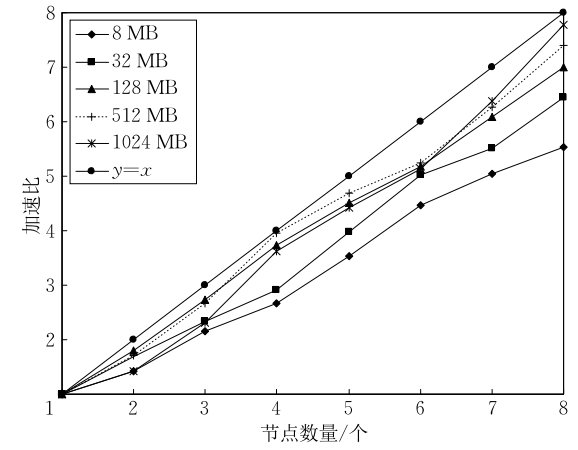


(c) 可扩展性

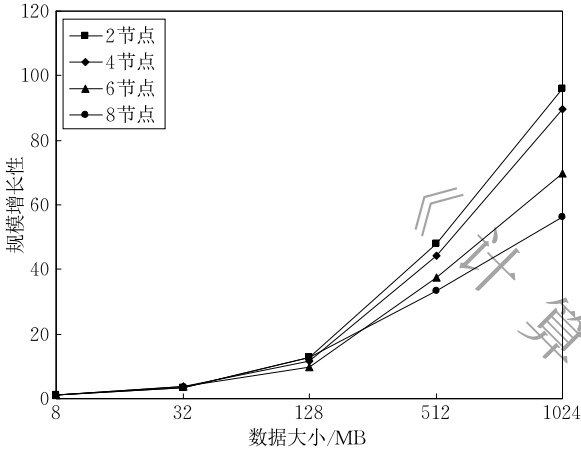
图 11　SparkR 运行 EMD 算法的并行指标

若有更大规模的数据分析任务时,可通过增加计算节点,来线性降低计算时间.值得注意的是,对于较小的数据集(如 8 M),EMD 算法在 LTSAF 上面的并行加速比较低,这主要是因为 Spark 集群启动耗时较长,经测试,空作业启动需要花费 7 s,而 LTSAF 计算绝对时间较短,如 8 节点计算 8 M 数据仅耗时 12 s,启动开销占比过高导致加速比下降.

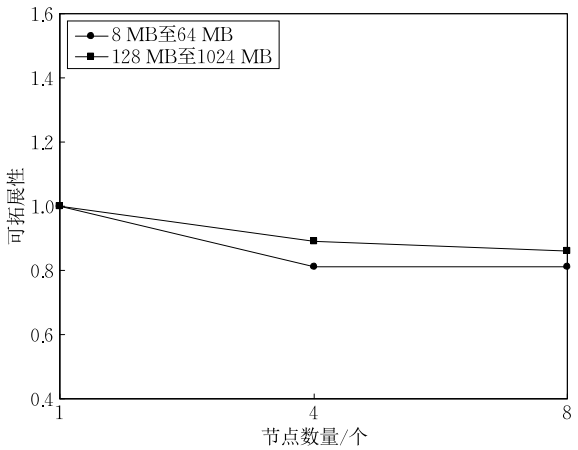
虽然直观上看 LTSAF 在 Speedup 指标上的表



(a) 加速比

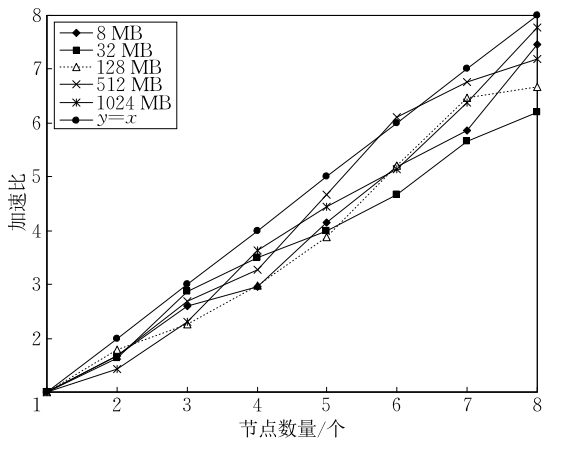


(b) 规模增长性

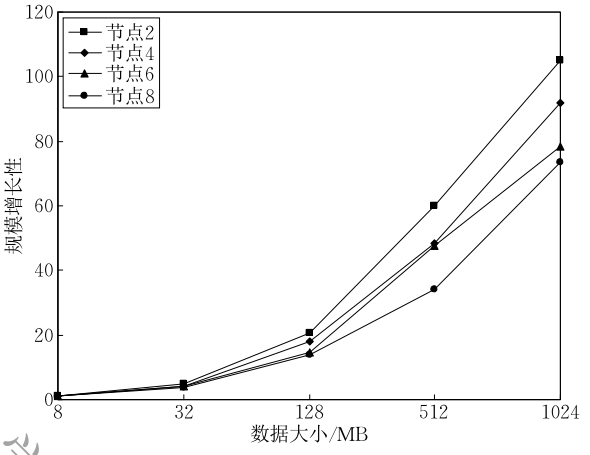


(c) 可扩展性

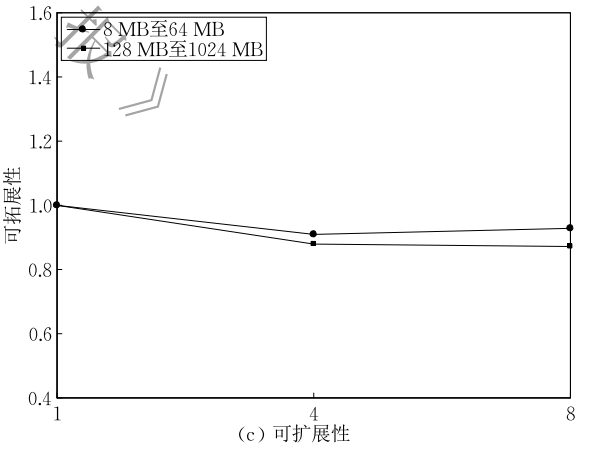
图 12 LTSAF 运行 SSA 算法的并行指标



(a) 加速比



(b) 规模增长性



(c) 可扩展性

图 13 SparkR 运行 SSA 算法的并行指标

现不及 SparkR,但其执行效率比 SparkR 较好. 原因是 SparkR 采用了 dataframe(数据帧)编程方式,无法显示控制 stage(阶段)和 DAG(有向无环图,Directed Acyclic Graph)图. 在 LTSAF 中,采用 Spark RDD(弹性分布式数据集,Resilient Distributed Datasets)进行编程,灵活控制 RDD 算子,尽量避免多余的宽依赖操作,从而生成尽量少的 stage,提高框架的执行效率.

5.5.2 规模增长性

EMD 和 SSA 算法的规模增长性 Sizeup 测试结果如图 10(b)～图 13(b)所示. 并行节点增多,LTSAF 与 SparkR 的规模增长相近,表现出良好的线性增长趋势.

5.5.3 可扩展性

EMD 和 SSA 算法的可扩展性 Scaleup 测试结果如图 10(c)～图 13(c)所示. 随着节点数目的增

加，二者 Scaleup 指标均保持在 0.8 至 1.0 之间. 完全数据并行可解除数据集规模对时间序列分析任务的制约，若数据量持续增加，计算时间可被较准确估计.

5.6　近似解正确性实验

LTSAF 生成的分析结果为近似解，而非精确解. 为了验证近似解的正确性，本文采用仿真时间序列和振动监测时间序列进行对比实验.

5.6.1　仿真时间序列

仿真时间序列的各个分量已知，作为精确解与近似解比较相似度. 如表 4 所示，LTSAF 生成的近似解与 RStudio 的精确解相似程度很高. 采用 EMD 算法进行序列分析，能量较高的第一分量和第二分量的近似程度超过 99.99%，能量较低的第三分量的相似度为 99.26%. 采用 SSA 算法进行序列分析时，所有三个分量与精确解的相似度超过 99.99%. SSA 算法的近似效果更好，但计算复杂度也更高.

表 4　仿真时间序列的正确性验证

EMD 相似度比较/%			SSA 相似度比较/%		
第一分量	第二分量	第三分量	第一分量	第二分量	第三分量
99.99	99.99	99.26	99.99	99.99	99.99

5.6.2　振动监测时间序列

振动监测时间序列是对高速列车振动实验台走行部不同部位的监测数据，共分为四种故障工况，分别是横向减振器故障、抗蛇行减振器故障、空气弹簧故障和无故障四种. 传感器采样频率 243 Hz，连续采集 60 s，每种工况包含长度为 14580 的时间序列. 由于监测数据是非线性非平稳的，无法精确获知其物理特性，本文采用文献[32]中基于 EMD 分解的故障分类方法对不同工况进行识别，通过对比 LTSAF 近似解与原始方法的分类效果来验证 LTSAF 的正确性.

本文采用与文献[32]相同的经验参数，默认分段长度 $S=256$，周期 $T=11$. 采用 EMD 算法和 SSA 算法分析监测时间序列，训练分类器. 测试了三种不同的分段长度 192，256 和 512. 由于不同算法处理近邻数据的方式不尽相同，所以冗余范围取值受算法特点影响较大，而与数据本身关系不大. 对于 EMD 算法，采用图 8 中冗余范围 $3T$. 为了简便起见，SSA 算法的冗余范围同样取值 $3T$. 其中 EMD 算法中第二组无冗余的情况是文献[32]的分类结果，此时分段长度 256，冗余范围为 0.

如图 14 所示，故障识别效果受分段长度影响较大，文献[32]中的经验值 256 并不是最佳取值，EMD 算法在分段长度 192 取得最好效果，这是由于原始序列较短，更短的分段能产生更多的学习样本，进而提高识别率. 分段长度取 512 时，学习样本太少，识别率远远低于另外两种情况. 采用 SSA 算法进行故障识别的结果与 EMD 类似，其中较好的分段长度为 256. 考虑到 SSA 算法复杂度远远高于 EMD 算法，高铁故障识别多采用 EMD 算法进行序列分解[32].

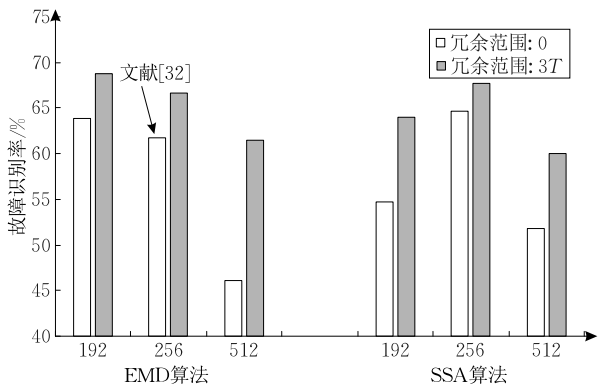


图 14　振动监测时间序列的故障识别率

值得注意的是，冗余方案可以提高故障识别率，且表现十分稳定，平均提升效果为 14.5%. 同时冗余方案缓解了不同对照组的识别率差异，使故障识别率稳定在 60%以上. 当分段长度不确定时，仍然可以取得较好的识别效果，保证并行分析算法近似解的正确性.

6　总　结

近年来，工业大数据的蓬勃发展使得时间序列分析不再是单纯的数据分析问题，而更多地是一个面向大规模数据的复杂系统问题. 现有的并行分析算法数量有限，常与平台相互绑定，可扩展性较差. 本文设计了一种通用的近似解分析框架，支持第三方算法快速实现并行化，解决因数据规模过大而导致的算法适用性问题.

根据时间序列的访存模式呈现局部性特点，本文研究了算法并行化中任务划分问题，在无法获得精确解的情况下采用数据并行的计算方式求取近似解. 本文从任务划分、时空优化、参数求解等角度介绍了大规模时间序列分析框架 LTSAF 的设计思想. 同时以 Spark 平台为例实现了 LTSAF 原型系统，展

示了并行参数的提取过程,验证了该原型系统可以保证分析结果的正确性,并能有效提高序列分析的处理速度和规模.由于 LTSAF 实现完全数据并行,并行单元之间不需要通信,如果处理数据量持续增加,LTSAF 的运算时间可以被准确估计.

总体来说,LTSAF 通过分治冗余维护了时间序列的数据关联特性,保证了近似分析结果正确性,解除了分析算法受限于数据规模的问题,实现加速能力可以随着并行度增加保持近似线性的增长.同时 LTSAF 提供了多样的算法扩展方式,不依赖于特定计算引擎和平台,可以大大缩短并行算法的开发时间.

大规模时间序列分析框架还存在需要进一步研究和改善的技术问题.例如对某些局部性较差的分析算法,数据并行模型仅通过冗余无法满足分析精度的要求.另外,原型系统需要加强数据分区、缓存等手段优化并行机制,使并行指标有更高的提升.

参 考 文 献

[1] Field A. Discovering Statistics Using IBM SPSS Statistics. Los Angeles, USA: Sage Publications Ltd, 2013

[2] Miškuf M, Michalik P, Zolotová I. Data mining in cloud usage data with Matlab's statistics and machine learning toolbox//Proceedings of the International Symposium on Applied Machine Intelligence and Informatics. Herl'any, Slovakia, 2017: 377-382

[3] Meddah I H, Belkadi K. Parallel distributed patterns mining using Hadoop MapReduce framework. International Journal of Grid and High-Performance Computing, 2017, 9(2): 70-85

[4] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets//Proceedings of the USENIX Conference on Hot Topics in Cloud Computing. Boston, USA, 2010: 10-18

[5] Huang Yi-Hua. Research progress on big data machine learning system. Big Data Research, 2015, 1(1): 28-47(in Chinese)
(黄宜华. 大数据机器学习系统研究进展. 大数据, 2015, 1(1): 28-47)

[6] Wu Xin-Dong, Ji Sheng-Wei. A comparative study on Map-Reduce and Spark for big data analytics. Journal of Software, 2017, 28(7): 1770-1791(in Chinese)
(吴信东, 嵇圣硙. MapReduce 与 Spark 用于大数据分析之比较. 软件学报, 2017, 28(7): 1770-1791)

[7] Giacomelli P. Apache Mahout Cookbook. Birmingham, UK: Packt Publishing, 2013

[8] Boehm M, Reinwald B, Hutchison D, et al. On optimizing operator fusion plans for large-scale machine learning in SystemML. Proceedings of the VLDB Endowment, 2018, 11(12): 1-14

[9] Hellerstein J M, Ré C, Schoppmann F, et al. The MADlib analytics library or MAD skills, the SQL//Proceedings of the Very Large Data Bases Endowment. Istanbul, Turkey, 2012: 1700-1711

[10] Meng X, Bradley J, Yavuz B, et al. MLlib: Machine learning in apache spark. Journal of Machine Learning Research, 2015, 17(1): 1235-1241

[11] Mervyn G M, William J K. SAS for Data Analysis: Intermediate Statistical Methods. New York, USA: Springer, 2008

[12] Greenfield T, Metcalfe A. Design and Analysis Your Experiment Using Minitab. UK: Hodder Arnold, 2007

[13] Herbrich R. Machine learning at Amazon//Proceedings of the 10th ACM International Conference on Web Search and Data Mining. Cambridge, UK, 2017: 535

[14] Zhang Peng, Duan Lei, Qin Pan, et al. Top-k contrast sequence pattern mining based Spark. Computer Research and Development, 2017, 54(7): 1452-1464(in Chinese)
(张鹏, 段磊, 秦攀等. 基于 Spark 的 Top-k 对比序列模式挖掘. 计算机研究与发展, 2017, 54(7): 1452-1464)

[15] Yan Yu-Liang, Dong Yi-Hong, He Xian-Mang, et al. FSMBUS: A frequent subgraph mining algorithm in single large-scale graph using Spark. Journal of Computer Research and Development, 2015, 52(8): 1768-1783(in Chinese)
(严玉良, 董一鸿, 何贤芒等. FSMBUS: 一种基于 Spark 的大规模频繁子图挖掘算法. 计算机研究与发展, 2015, 52(8): 1762-1783)

[16] Chen Jianguo, Li Kenli, Rong Huigui, et al. A periodicity-based parallel time series prediction algorithm in cloud computing environments. Information Sciences, 2018, 12(4): 35-45

[17] Bhatia V, Rani R. Ap-FSM: A parallel algorithm for approximate frequent subgraph mining using Pregel. Expert Systems with Applications, 2018, 16(106): 217-232

[18] Mishra S, Lee Y C, Nayak A. Distributed genetic algorithm on GraphX//Proceedings of the Advances in Artificial Intelligence. Hobart, Australia, 2016: 548-554

[19] Wei J, Chen K, Zhou Y, et al. Benchmarking of distributed computing engines spark and GraphLab for big data analytics //Proceedings of the 2nd International Conference on Big Data Computing Service and Applications. Oxford, UK, 2016: 10-13

[20] Spielman D A, Teng S H. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. SIAM Journal on Computing, 2013, 42(1): 1-26

[21] Ballard G, Druinsky A, Knight N, et al. Hypergraph partitioning for sparse matrix-matrix multiplication. ACM Transactions on Parallel Computing, 2016, 3(3): 1-34

[22] Carlini E, Dazzi P, Esposito A, et al. Balanced graph partitioning with apache spark. Euro-Par Lecture Notes in Computer Science, 2014, 12(5): 65-75

[23] Yang S, Yan X, Zong B, et al. Towards effective partition management for large graphs. SIAM Journal on Computing, 2012, 12(3): 2-34

[24] Ho Q, Cipar J, Cui H, et al. More effective distributed ML via a stale synchronous parallel parameter server. Advances in Neural Information Processing Systems, 2013, 12(2013): 1223-1231

[25] Lu X Y, Shi H Y, Javed M H, et al. Characterizing deep learning over big data (DLoBD) stacks on RDMA-capable networks//Proceedings of the High-Performance Interconnects. Santa Clara, USA, 2017: 87-94

[26] Zhu Hu-Ming, Li Pei, Jiao Li-Cheng, et al. Review of parallel deep neural network. Chinese Journal of Computers, 2018, 41(8): 1861-1881(in Chinese)
(朱虎明，李佩，焦李成等. 深度神经网络并行化研究综述. 计算机学报，2018，41(8)：1861-1881)

[27] Prajapati V. Big Data Analytics with R and Hadoop. Birmingham, UK: Packt Publishing, 2013

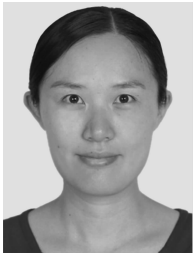
[28] Venkataraman S, Yang Z H, et al. SparkR: Scaling R programs with spark//Proceedings of the 2016 International Conference on Management of Data. San Francisco, USA, 2016: 1099-1104

[29] George L. HBase: The Definitive Guide: Random Access to Your Planet-Size Data. USA: O'Reilly, 2011

[30] Qin Na, Jin Wei-Dong, Huang Jin, et al. Ensemble empirical mode decomposition and fuzzy entropy in fault feature analysis for high-speed train bogie. Control Theory & Applications, 2014, 31(9): 1245-1251(in Chinese)
(秦娜，金炜东，黄进等. 高速列车转向架故障信号的聚合经验模态分解和模糊熵特征分. 控制理论与应用，2014，31(9)：1245-1251)

[31] Yamazaki I, Wu K, Simon H. Nu-TRLan user guide version 1.0: A high-performance software package for large-scale harmitian eigenvalue problems. Lawrence Berkeley National Laboratory, 2008, 2008(1): 3-27

[32] Chen Zhi, Li Tian-Rui, Li Ming, Yang Yan. Fault diagnosis method of high-speed rail based on compute unified device architecture. Journal of Computer Applications, 2015, 35(10): 2819-2823(in Chinese)
(陈志，李天瑞，李明，杨燕. 基于计算统一设备架构的高铁故障诊断方法. 计算机应用，2015，35(10)：2819-2823)



TENG Fei, Ph.D., associate professor. Her research interests include parallel computing, cloud computing, etc.

HUANG Qi-Chuan, M. S. His research interests include cloud computing, Internet technology, etc.

LI Tian-Rui, Ph. D., professor. His research interests include data mining and knowledge discovery, cloud computing, granular computing and rough sets, etc.

WANG Chen, Ph. D., professor. His research interests include industrial big data, data platform technology, etc.

TIAN Chun-Hua, Ph. D., professor. His research interests include data analysis and operation optimization, etc.

Background

With the development of industrial big data, large-scale time series analysis is not only an analysis problem, but also a complicated system problem. Time series data generated by sensors has led a great performance on a variety of problems, such as Internet of Things, Industrial Internet, and Edge Computing etc. However, with the increasing efficiency requirements for the practical reasons, the size of time series becomes explosively large scale, real-time time series computing is inefficient or even invalid with traditional analysis tools such as R and Matlab. Therefore, it is necessary to accelerate the computing large-scale time series in parallel. Although there are some commercial or open-source software for distributed computing, most of them only provide a few parallel algorithms for machine learning. These algorithms are platform-dependent, that is to say, the same algorithm should be developed repeatedly for different platforms. Since R and other third party libraries contain a great quantity of algorithms for matrix manipulation, it becomes important to paralyze the analyzing large-scale time series with credible third-party libraries and distributed platforms.

This work is partially supported by the National Key Research and Development Program of China

(No. 2018YFB1701502), and the Sichuan Science and Technology Program (No. 2019YJ0214). Our research team has been working on high performance computing and massively parallel machine learning for many years. Related works have been published in international journals and conferences, such as TPDS, TC, JoS, HPCC, etc.

In this paper, we give a brief review of parallel computing time series in the past, and point out that intermediate data exchanging becomes the key constraint for parallel computing. For the first time, we propose an approximate solution in the case that the exact solution can not be obtained. We design a general framework that can increase the processing size of large data sets and can transplant the verified algorithms from other third-party libraries. Based on the insight of divide and conquer, the framework analyzes independent segmentations of original time series on a distributed storage and processing environment, and then concatenates partial results to generate the final result of acceptable precision. We model the time-space optimization problem to obtain the best length of segmentation. We also develop a prototype Spark system, and validate the accuracy and efficiency with large-scale time series data set.