Hans Journal of Data Mining 数据挖掘, 2016, 6(4), 158-167
Published Online October 2016 in Hans. http://www.hanspub.org/journal/hjdm
http://dx.doi.org/10.12677/hjdm.2016.64018

Hans汉斯

Research and Implementation of Parallel Data Mining of Process Object Based on Spark

Yafei Zheng[1], Tao Du[1], Lianjiang Zhu[2], Shouning Qu[1,2]

[1]School of Information Science and Engineering, University of Jinan, Ji'nan Shandong
[2]Information Network Center, University of Jinan, Ji'nan Shandong
Email: eo_dut@ujn.edu.cn

Received: Sep. 30th, 2016; accepted: Oct. 19th, 2016; published: Oct. 24th, 2016

Abstract

In this paper, we study the parallel data mining based on Spark, and apply it to the data analysis of process object. We propose some parallel algorithm flow solutions based on Spark by studying the algorithm flow of stand-alone process object data mining. Through programming, parallel efficiency testing and algorithm tuning, we conclude an optimized parallel algorithm flow. These solutions improve the computational efficiency.

Keywords

Data Mining, Parallel Computing, Process Object, Spark, MapReduce

基于Spark的流程对象并行数据挖掘的研究与实现

郑雅飞[1]，杜　韬[1]，朱连江[2]，曲守宁[1,2]

[1]济南大学信息科学与工程学院，山东 济南
[2]济南大学信息网络中心，山东 济南
Email: eo_dut@ujn.edu.cn

收稿日期：2016年9月30日；录用日期：2016年10月19日；发布日期：2016年10月24日

文章引用: 郑雅飞, 杜韬, 朱连江, 曲守宁. 基于 Spark 的流程对象并行数据挖掘的研究与实现[J]. 数据挖掘, 2016, 6(4): 158-167. http://dx.doi.org/10.12677/hjdm.2016.64018

摘　要

本文研究了基于Spark的并行数据挖掘，并将其应用到了流程对象数据分析中。文章通过对串行的流程对象数据挖掘算法流的研究，提出了一种基于Spark并行计算框架的并行化算法流解决方案，并通过编程实现、并行效率测试、算法调优，最终得出一个并行效果良好的并行数据挖掘方案。该并行方案明显提高了计算效率。

关键词

数据挖掘，并行计算，流程对象，Spark，MapReduce

1. 引言

分布式控制系统(Distributed Control System, DCS)在流程工业中应用广泛，其依靠分布于各个环节的传感器实时检测得到的数据进行系统控制。一方面，长期的检测过程可以得到大量数据。另一方面，由于计算机技术的空前发展，尤其是硬件存储技术以及数据库技术的发展，使得海量数据的存储成为可能。

信息时代数据的爆炸式增长，让人们愈发认识到数据的重要性，产生于上世纪 90 年代的数据挖掘(Data Mining, DM)技术受到人们广泛关注。数据挖掘就是从大量的、不完全的、有噪声的、模糊的、随机的实际应用数据中，提取隐含在其中的、人们事先不知道的、但又是潜在有用的信息和知识的过程。数据挖掘为企业决策与控制提供了数字化的指导信息，使得决策与控制不再过于盲目。数据挖掘在工业过程控制中的研究也愈发火热。

与数据挖掘一起，受到人们关注的是大数据处理(Big Data Processing)技术。并行计算(Parallel Computing)便是大数据处理的首选，并行计算通过多处理机同时处理同一事务进而加速计算过程。现有的并行计算机主要有 SIMD 架构和 MIMD 架构，在并行计算机上同样存在着多种并行编程模型和架构，常用的主要有 MPI、OpenMP、Hadoop、Spark [1]-[3]等，这些并行编程模型和架构在并行化机理上不尽相同，有的是基于数据划分机制的并行，有的是基于任务划分机制的并行，因此适用的场景也不尽相同。在这些并行计算框架中，Spark [3] [4]是当前备受人们关注的开源集群计算框架，Spark 基于改进的 MapReduce [5]，计算效率较 Hadoop 有明显提高。

然而，设计并行计算程序比设计串行程序具有更多的挑战性。针对具体的并行计算模型，并不是任何串行算法都可以轻而易举地转换为并行算法。针对不同的问题，往往要经过仔细的考量之后才能得到一个可以实施且效果良好的并行计算方案。

本文在 Spark 基础之上设计并实现了并行化的流程对象数据挖掘算法，主要包含了数据预处理、时序调整、环节聚类、关系分析以及状态关联这几个环节，最终实验表明该并行化设计是可行的，并且有了较大的效率提升。

2. 相关理论研究

2.1. 流程对象模型

2.1.1. 流程工业检测过程

在流程工业的分布式控制系统中，生产过程由一个个环节构成，每个环节通常需要加装许多检测装置，通过数据接口，将检测数据传入数据库/数据仓库中存储起来，用于监测过程状态[1]。我们可以将流程工业生产过程数据检测结构描述为图 1。

在集中数据库中存储着各个环节的检测数据，形式如表 1 所示。

2.1.2. 流程对象模型定义

基于上述介绍和相关研究[1]，我们将流程工业生产过程所具有的一般特征列举如下：

1) 多环节：可以把整个生产流程看作一个开环系统，局部的相对独立的生产步骤看作闭环系统。多个环节前后相连，构成一个流程。

2) 时序性：生产是从一个环节流向一个环节，也就具有时序性。时序性主要有以下含义：

① 前后顺序。

② 环节延迟。

3) 单向相关性：生产过程中，前一环节的输出作为后一环节的输入，后一环节依赖于前一环节，反过来则没有。

根据上述特性，我们可以抽象出：流程对象(Process Object, PO)是由多个环节构成的具有单向向后相关性和时序性的整体，其所有环节组成了一个时间序列集合，这个时间序列集合中的数据来自各个环节的可采样检测点，检测点采样在时间 $T = \{t_1, t_2, \cdots, t_m\}$ 内完成，其中 $t_1 < t_2 < \cdots < t_m$，则流程对象 $\mathcal{X}$ 可以按如下公式表示：

$$\begin{aligned}\mathcal{X} = \{ & X_1(x_1(t_1), x_1(t_2), \cdots, x_1(t_m)), \\ & X_2(x_2(t_1), x_2(t_2), \cdots, x_2(t_m)), \\ & \cdots \\ & X_n(x_n(t_1), x_n(t_2), \cdots, x_n(t_m))\}\end{aligned}$$

其中 o， $X_i, X_j (i, j = 1, 2, \cdots, n)$ 为在时间 $T = \{t_1, t_2, \cdots, t_m\}$ 内的可采样环节数据。

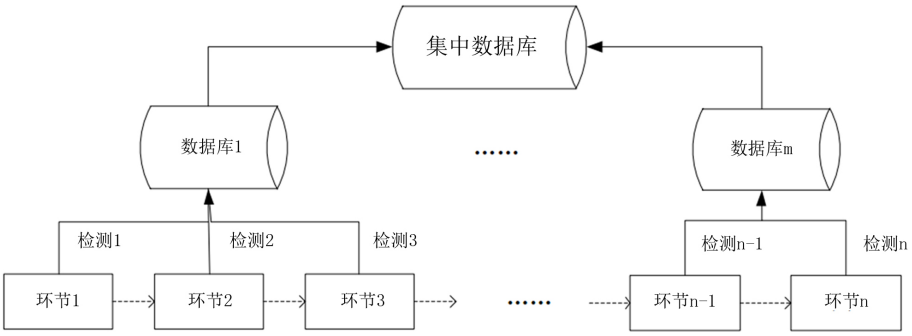

Figure 1. Detection structure of process industry production process
图 1. 流程工业生产过程检测结构

Table 1. Detection data of links
表 1. 环节检测数据

检测时间	环节 1	环节 2	环节 3	…	环节 n − 1	环节 n
T_1	X_{11}	X_{21}	X_{31}	…	$X_{(n-1)1}$	X_{n1}
T_2	X_{12}	X_{22}	X_{32}	…	$X_{(n-1)2}$	X_{n2}
T_3	X_{13}	X_{23}	X_{33}	…	$X_{(n-1)3}$	X_{n3}
…	…	…	…	…	…	…
T_m	X_{1m}	X_{2m}	X_{3m}	…	$X_{(n-1)m}$	X_{nm}

2.2. 集群计算框架 Spark

Apache Spark 是目前比较流行的并行计算框架，它基于改进的 MapReduce 模型。在经典的 MapReduce 模型中，数据通常被分区存储在不同的计算节点上，计算也被分为 Map 和 Reduce 两个阶段，其中 Map 阶段负责对分区数据进行转换，而每一次 Map 的计算结果都会被缓存到磁盘上进行容错，Reduce 阶段负责将 Map 阶段的计算结果整合规约到一块并得到最终结果，而在 Map 和 Reduce 中间还可能存在 Shuffle 过程。Spark 对于经典 MapReduce 的改进在于，Spark 将计算的中间结果保存在内存中，而不是缓存到磁盘中，因此相较于经典 MapReduce，Spark 具有更高的并行执行效率且更适合于数据挖掘和机器学习等需要迭代处理的领域[6]。

与 MapReduce 一样，Spark 也是基于数据并行的。Spark 将数据集抽象为 RDD [7]，RDD 中数据以分区的形式分布在不同节点上，对 RDD 的操作就被转换为对数据集的并行操作。

3. 基于 Spark 的并行化设计

为了挖掘流程对象各个环节之间的相互影响关系，基于串行的流程对象知识发现算法流[8]，本文提出了如图 2 所示的并行化算法流。

从图 2 可以看出，算法流主要包括数据预处理、时序调整、环节聚类、关系分析以及状态关联这几个环节。下面分别介绍每个环节的并行化设计。

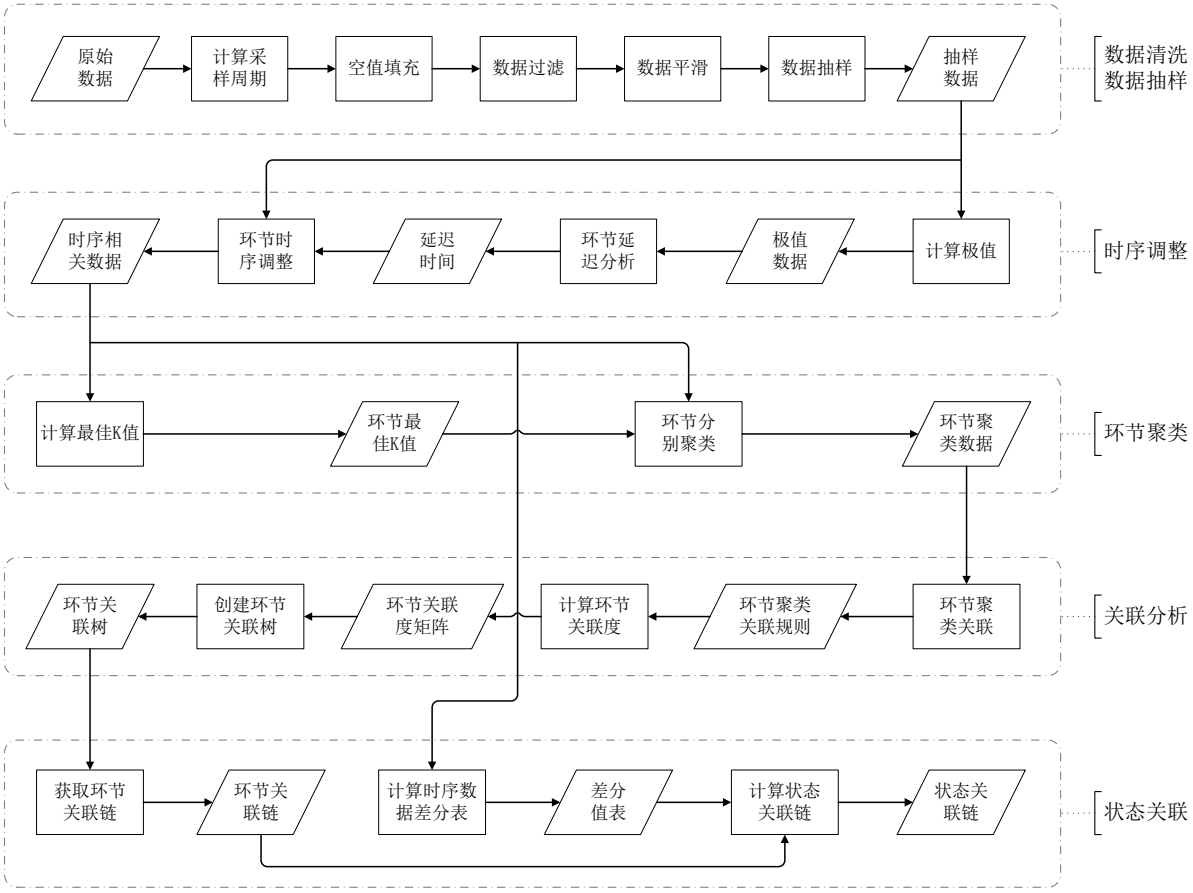

Figure 2. Algorithm flow of knowledge discovery system of process object
图 2. 流程对象知识发现系统算法流

3.1. 边界数据处理

受数据并行的计算模型所限，对于数据之间需要协同计算的情况，处理起来显得不是很灵活。差分法空值填充算法和数据平滑算法都属于这种类型，在计算当前数据的同时，也同时需要前后相关联的数据。单纯的基于数据划分的并行不能适用于此类型的计算任务。

为了解决上述问题，我们需要在计算当前数据分区的同时，也拥有当前计算所需要的其他分区的数据。在 Spark 中，我们可以将所有数据通过广播的形式复制到所有计算节点，但是巨大的通信开销会导致效率极其低下。为此，本文通过获取分区边界处必要的少数几条记录，并对其进行广播，这样大大减少了数据广播所带来的通信开销。

对于一个分区，我们通常需要获取分区头部和尾部的边界数据。下面给出分区头部边界数据获取算法：

输入：分区数据 p，需获取的边界记录数目 n

输出：满足要求的最少记录数组 a

```
1: if n< 0 then
2: links = Φ
3: for record from p.head to p.last if links.length<linkcount do
4: for i from 1 to record.length do
5: if record[i] not null &&i not in links then
6: links += i
7:     if record not in a then
8: a += record
9: endif
10: endif
11: endfor
12: endfor
13: else
14: for j from 1 to n do
15: a += p[j]
16: endfor
17: endif
18: return a
```

对图 3 所示分区尾部(其中，每一列代表一条记录，每一个单元格代表一个环节数据，白色单元格为空缺值)使用上述算法，当输入数据小于 0 时，我们将得到如图 4 所示的边界数据(其中，深灰色标记的记录为最终获取的边界数据)。

在该并行化系统中，边界数据处理是较为频繁的操作，减少边界数据处理的通信开销将会大大加速并行计算过程。

3.2. 数据预处理

数据预处理将原始数据转换为方便后续算法进行分析的数据，通常在数据挖掘和知识发现中占比较大的比重。在本算法流中，数据预处理部分也占有较大比重，大致可以分为：数据清洗、数据抽样、时

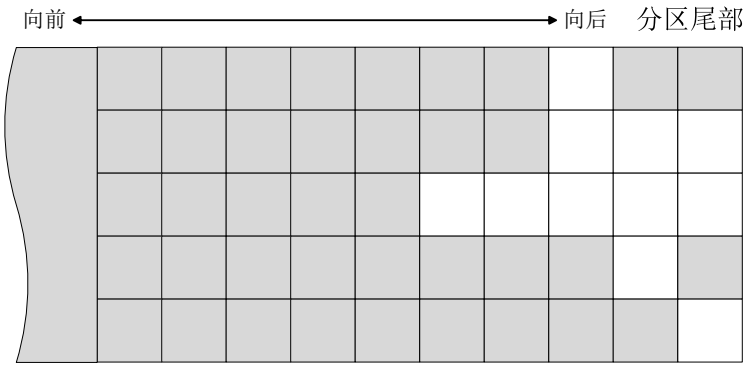

Figure 3. Partition boundary data processing
图 3. 分区边界数据处理示意图

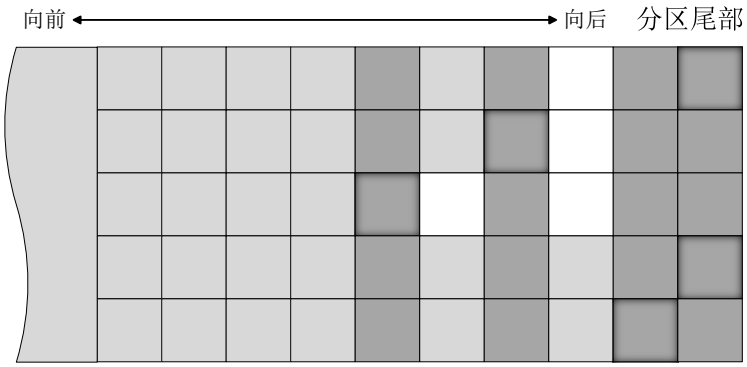

Figure 4. Partition boundary data processing
图 4. 分区边界数据处理示意图

序调整。

3.2.1. 数据清洗

数据清洗的步骤主要包括：空值填充、数据过滤以及数据平滑。

空值填充：在采样数据记录中，可能存在空缺环节数据，我们采取填充的方法来进行处理。空值填充使用等差填充法，首先获取空值前后第一个非空值，然后按照等差数列的分布逐个填充空缺值。在 Spark 中，处理这种需要前后关联起来一块处理的数据时存在一定的困难。在此，我们需要使用前面介绍的边界数据处理算法，获取从分区边界开始的各个环节第一个非空值所在的记录，并将所有环节的边界数据收集到 driver 节点，然后广播到各个计算节点。

数据过滤：空值填充之后的数据中仍可能存在空缺记录，空缺记录严重影响后续的时序计算，在本系统中，数据过滤和数据抽样保证了抽样数据中不再含有空缺记录。在 Spark 中，对数据的划分(即 RDD 分区)是大尺度的，因此我们在数据过滤时针对 RDD 分区进行。对于在数据分区内部含有空缺记录的分区，我们将其过滤掉，这同时也减小了后续计算的数据量，从而加速了计算。

数据平滑：检测数据中不免会存在噪声，为了削减噪声数据所带来的计算误差，我们需要对数据进行降噪处理。在本系统中，我们使用均值平滑的方式对环节数据进行平滑，滑动窗口大小为 3，步进为 1。在 Spark 中，这同样需要先将每个分区边界处的 1 条数据广播到所有节点。

3.2.2. 数据抽样

为了加速计算过程，我们需要对数据进行抽样。在本算法流中，抽样的作用主要有 3 点：1) 减少数

据量，加速计算；2) 去除含有空缺记录的数据分区，使用于计算的数据连续；3) 获取数据中最具有挖掘价值的数据段。

在本文中，我们使用基于滑动窗口的数据变化量最大化抽样方法，最终获取原始数据中的一段连续的且数据变化量最大的数据，数据变化量即是相邻数据之间差值的绝对值之和。对于串行的数据抽样算法，数据划分级别为记录，算法中的滑动窗口可以设置为大小等于抽样量，步进为 1，然后就可以使用滑动窗口来计算所有数据，从而得到最佳抽样数据段。

然而数据分区不可能仅包含一条记录，也不是包含很少几条记录，为了达到较好的计算效率，每个数据分区通常会包含大量数据。因此，在基于 Spark 的并行化设计中，我们采用近似的大步幅的滑动窗口算法，滑动窗口步进为 1 个数据分区。首先计算出所有数据分区的单独的数据变化量，然后将分区编号并收集到 driver 节点，在数据过滤阶段被滤掉的分区作为空缺分区也被编号。如图 5 所示，灰色方格为从各个计算节点收集得到的数据分区的变化量，白色方格代表被过滤掉的空缺分区，所有分区依次排列并被编号。空缺分区前后为不同的数据段，在窗口滑动的过程中遇到空缺分区将成为无效窗口，图 6 为有效窗口，图 7 窗口中包含了空缺分区，为无效窗口，窗口需继续滑动，直到不包含空缺分区为止。

容易看到该近似抽样算法最终的抽样量和需要的抽样量可能相差较大，为了使其更加接近，我们最好将数据分区的大小设置为抽样量的 1/n (n 为整数)。

3.2.3. 时序调整

抽样之后的数据仍存在一个问题：环节数据在时序上不匹配，也即同一条记录中的各个环节数据之间并不是直接相互影响关系，因为环节之间存在时间延迟，因此我们需要找出环节之间的延迟时间，并对数据进行时序调整。时序调整主要分为两步：环节延迟分析和环节数据调整。

环节延迟分析用于找出流程对象中前后两环节产生影响所需要的延迟时间，下面给出其并行设计的算法：

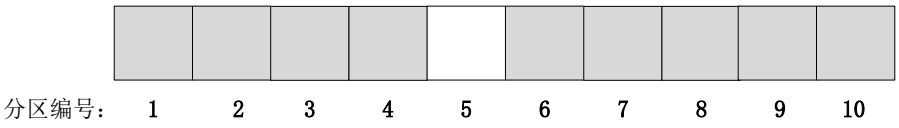

Figure 5. Number of data partition
图 5. 数据分区编号

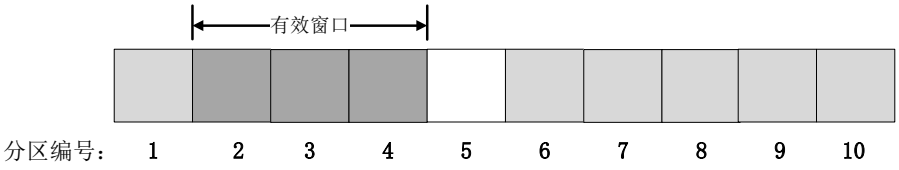

Figure 6. Valid window
图 6. 有效窗口

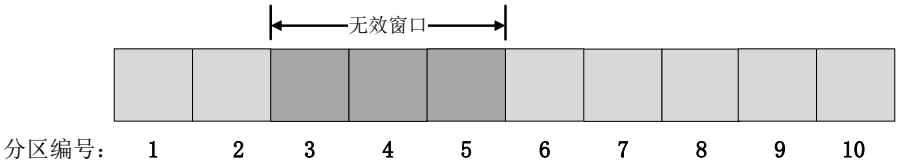

Figure 7. Invalid window
图 7. 无效窗口

输入：抽样数据中所有环节的极值点对应的时间的集合 e，环节集合 l，基准环节 x

输出：各环节相对于基准环节 x 的延迟时间数组 a

1: *links* = {*x*}

2: *a* = {((*x*, *x*), 0)}

3: **for***y* in *l***if***y* not in *links***do**

4: *distance* = Φ

5: **for***extreme_x* in *e*[*x*]**do**

6: *tmp* = Φ

7: **for***extreme_y* in*e*[*y*] **do**

8: *tmp*+= *extreme_y* – *extreme_x*

9: **endfor**

10: *distance* += value of *tmp*.min_abs_value

11: **endfor**

12: *a* += ((*x*, *y*), *distance*.mode_value)

13: **endfor**

14: **return***a*

上述算法得出的环节相对延迟时间可能为正，可能为负，也可能为零。

为了加速计算并使其易于并行，在计算极值点时，我们采取二元矩阵的形式存储极值点数据，如果某一环节数据在该环节中属于极值点则将其置为 1，否则置为 0。这种数据存储形式虽然空间复杂度较大，但是它保留了各环节数据之间的相对位置。基于这种数据存储形式，当我们需要计算基准环节 x 的一个极值时间点 m 到环节 y 所有极值时间点之间的距离绝对值最小值的时候，可以直接从极值点 m 所在记录开始向前向后扫描最近的环节 y 极值点即可，达到了快速定位的目的。

3.3. 环节聚类

经过数据预处理之后的数据是直接时序相关的，但由于数据量还很大，如果直接进行关联分析，会导致产生的关联规则数量巨大并且价值密度较低。为了解决这个问题，我们需要减少关联规则前后项的状态量，聚类刚好可以解决这个问题。本文中环节聚类就是使用 K-Means 算法将每个环节的数据聚类为 K 个类别，其中，环节 K 值通过凝聚度和分离度的轮廓系数算法自动确定，且每个环节的 K 值均为单独计算。

环节聚类的总体策略是：使用给定的 K 值区间中的每一个 K，对环节数据进行聚类，并计算聚类轮廓系数，最大轮廓系数对应的聚类数即为最佳 K 值，使用最佳 K 值的聚类结果即为最佳聚类结果。其中主要包含两个算法：1) K-Means 聚类算法；2) 基于轮廓系数的最佳 K 值获取算法[9]。

在传统 K-Means 算法的基础上，并行化 K-Means 需要将聚类中心广播到各计算节点，并将计算节点的计算结果汇聚到 driver 节点，以重新计算聚类中心。计算迭代进行直到聚类中心不再变化或者达到了指定迭代次数。

基于轮廓系数的最佳 K 值获取算法的并行化，首先需要将每个聚类的聚类数据单独广播到各个计算节点，据此计算分区数据中每一个数据的轮廓系数，所有数据的轮廓系数的平均值即为整个聚类的轮廓系数。

3.4. 环节关联与状态关联

关联分析用于挖掘环节之间的相互影响关系，我们首先通过并行化 Apriori 算法[10]挖掘出两两环节

之间的相互影响规则，我们称之为二项关联规则，然后将二项关联规则按照关联度大小进行排序，最终形成由最强关联度和次强关联度对应的二项规则组成的关联树。

我们将环节的状态分为上升、下降和不变三类，将状态应用于关联链中便成了状态关联链。状态关联链反映了不同环节之间状态变化的相互影响关系，通常情况下，一条关联链可生成多条状态关联链。通过遍历环节关联树可以得到环节关联链，有了环节关联链和环节状态变化表，只需进行简单的统计分析即可得出状态变化关联链。

4. 实现与优化

基于本文提出的并行化算法流，我们使用 Spark 实现了该知识发现系统，系统的框架图如图 8 所示。

在针对如表 1 所示的 10 万条数据上，我们对原始串行计算系统和基于 Spark 的并行计算系统做了性能对比。如表 2 所示，其中针对 Spark 并行系统的测试包含了本地运行测试和集群运行测试。

通过测试我们发现，该并行化计算系统较串行计算系统在计算效率方面有了较大提高。

5. 总结

Spark 集群计算具有易扩展、通用和高效的特点，基于 Spark 而设计的并行化流程对象知识发现系统在计算效率上明显优于串行算法。和其他并行计算算法设计一样，基于 Spark 的并行算法设计同样需要最大化地减少通信所带来的开销，本文的算法流设计充分体现了这一点。但由于算法策略的限制，并行

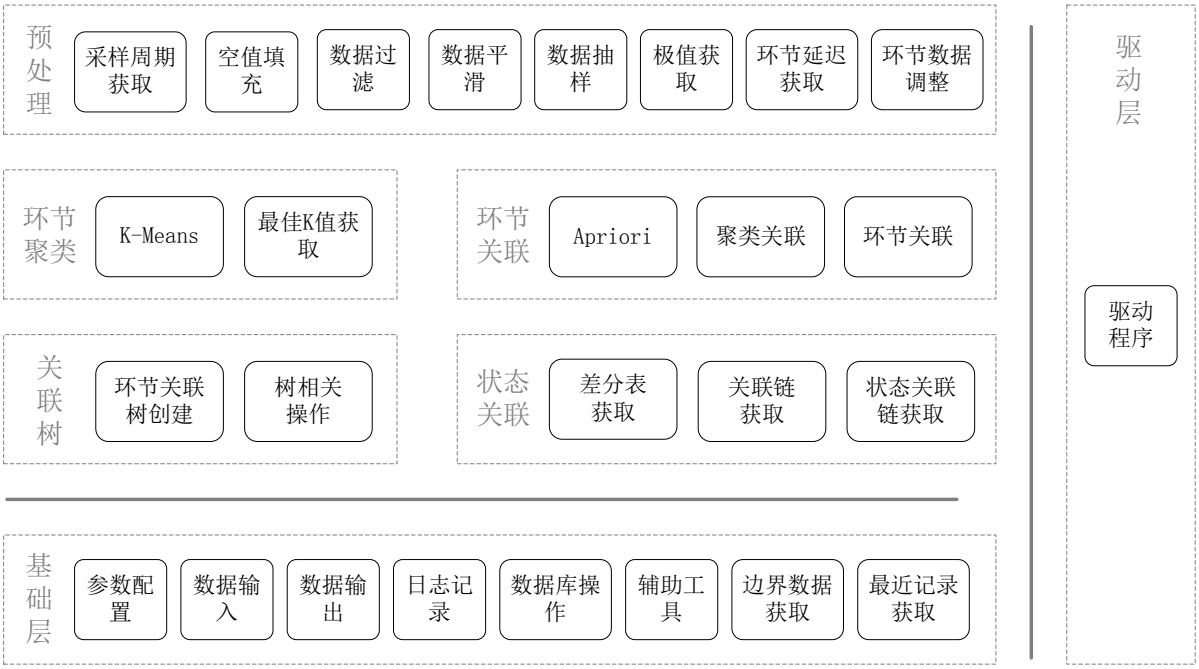

Figure 8. Framework of parallel knowledge discovery system
图 8. 并行化流程对象知识发现系统框架

Table 2. Execution efficiency comparison
表 2. 运行效率对比

	串行计算系统	并行计算系统(local)	并行计算系统(cluster)
运行环境	1 台：CentOS 7 (64 bit), 4GB RAM	1 台：CentOS 7 (64 bit), 4GB RAM	3 台：CentOS 7 (64 bit), 4GB RAM
运行时间	55 分 3 秒	57 分 47 秒	28 分 20 秒

化的流程对象知识发现系统还远没有达到实时处理的能力，下一步将研究增量实时处理，以使知识发现系统具有更高的实用价值。

参考文献 (References)

[1] Gropp, W., Lusk, E., Doss, N., *et al*. (1996) A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, **22**, 789-828. http://dx.doi.org/10.1016/0167-8191(96)00024-5

[2] Reyes-Ortiz, J.L., Oneto, L. and Anguita, D. (2015) Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf ☆. *Procedia Computer Science*, **53**, 121-130. http://dx.doi.org/10.1016/j.procs.2015.07.286

[3] Zaharia, M., Chowdhury, M., Franklin, M.J., *et al.* (2010) Spark: Cluster Computing with Working Sets. *Proceedings of the 2nd USENIX conference on Hot Topics in Cloud Computing*, USENIX Association, 10.

[4] Apache Spark. http://spark.apache.org

[5] Dean, J. and Ghemawat, S. (2004) MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of Operating Systems Design and Implementation* (*OSDI*), **51**, 107-113.

[6] Armbrust, M., Das, T., Davidson, A., *et al.* (2015) Scaling Spark in the Real World: Performance and Usability. *Proceedings of the VLDB Endowment*, **8**, 1840-1843. http://dx.doi.org/10.14778/2824032.2824080

[7] Zaharia, M., Chowdhury, M., Das, T., *et al.* (2012) Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, USENIX Association, 2.

[8] Song, Q., Guo, Q., Wang, K., *et al.* (2014) A Scheme for Mining State Association Rules of Process Object Based on Big Data. *Journal of Computer and Communications*, **2**, 17-24. http://dx.doi.org/10.4236/jcc.2014.214002

[9] Yu, H., Wen, J., Wang, H., *et al*. (2011) An Improved Apriori Algorithm Based On the Boolean Matrix and Hadoop. *Procedia Engineering*, **15**, 1827-1831. http://dx.doi.org/10.1016/j.proeng.2011.08.340

[10] 朱连江，马炳先，赵学泉．基于轮廓系数的聚类有效性分析[J]．计算机应用, 2010(S2): 139-141+198.