基于Hadoop的并行共享决策树挖掘算法研究

陈湘涛[1] 张 超[1] 韩 茜[2]
(湖南大学信息科学与工程学院 长沙 410082)[1] (美国莱特州立大学计算机科学与工程系 代顿 45435)[2]

摘 要 共享知识挖掘是指通过学习不同事物之间的共享知识,将学习到的知识应用到未知事物来加快认知未知事物。针对大数据集中串行共享知识挖掘算法效率低下的问题,结合云计算技术,提出了一种基于Hadoop的并行共享决策树挖掘算法(PSDT)。该算法采用传统的属性表结构实现并行挖掘,但其I/O操作过多,影响算法性能,为此,进一步提出了一种混合并行共享决策树挖掘算法(HPSDT)。该算法采用混合数据结构,在计算分裂指标阶段使用属性表结构,在分裂阶段采用数据记录结构。数据分析表明,HPSDT算法简化了分裂过程,其I/O操作是PSDT的0.34左右。实验结果表明,PSDT和HPSDT都具有良好的并行性和扩展性;HPSDT比PSDT性能更好,并且随着数据集的增大,HPSDT的优越性更加明显。

关键词 共享决策树,并行共享决策树,混合数据结构,云计算,Hadoop
中图法分类号 TP311 **文献标识码** A

Research on Parallel Shared Decision Tree Algorithm Based on Hadoop

CHEN Xiang-tao[1] ZHANG Chao[1] HAN Qian[2]
(Department of Information Science and Engineering, Hunan University, Changsha 410082, China)[1]
(Department of Computer Science and Engineering, Wright State University, Dayton 45435, USA)[2]

Abstract Shared knowledge is focused on the problems of knowledge discovery shared by two(or more) applications/datasets, and it can help users investigate a poorly understood dataset from a well understood dataset by analogy and transferring knowledge. However, with the advent of the era of big data, the existing algorithms based on serialized can not able to meet the need of the rapid data growth and the serial algorithm has low efficiency in big datasets. The Parallel Shared Decision Tree algorithm(PSDT) based on Hadoop was introduced by using cloud computing. The traditional attributes list structure is uses in the PSDT. But the overmuch operations of I/O occur in the parallel processes of this algorithm, which will influence the algorithm's performance. A new Hybrid Parallel Shared Decision Tree algorithm (HPSDT), which uses hybrid data structures, was proposed. The algorithm applies attributes-list to compute the split indicators parallelly, and data records-structure to split in the splitting procedure. Data analysis indicates that the algorithm of HPSDT simplifies the splitting process, and its operations of I/O are 0.34 times of the PSDT. The experimental results show that PSDT and HPSDT have good parallelism and scalability, furthermore, the performance of HPSDT is better than that of PSDT. Especially, the superiority is even more obvious with the increase of the dataset size.

Keywords Shared decision tree, Parallel shared decision tree, Hybrid data structure, Cloud computing, Hadoop

1 前言

在现实生活中,人们经常寻找不同事物、不同应用之间的共性。例如,从不同事物中总结规律,学习过程中举一反三,依靠物种相似性推测新发现细菌的生物结构等。为此,Dong等人[1]提出了共享的思想,即挖掘数据集之间共享的知识,并给出了一种共享决策树挖掘算法(SDT)。SDT算法同时挖掘已知数据集和未知数据集,建立共享决策树,从而学习二者的共性,快速地理解未知数据集,进而关注未知数据集的特性。

但是,SDT算法是基于传统的串行思想,只能处理小规模的数据,面对大规模数据时挖掘效率低下,构造决策树的时间会很长,甚至受内存限制无法运行。而共享决策树通常需要挖掘大量的数据才能获得有效的共享知识,尤其是在生物医学领域进行生物物种、基因序列、蛋白质功能或者疾病机理推测时,需要处理的数据通常都是大规模的,甚至是超大规模的。为此,本文提出一种并行的共享决策树挖掘算法来实现大数据集之间共享知识的挖掘。该方法通过引入并行决策树的思想和云计算技术,使得并行共享决策树挖掘算法能够在大规模计算集群上处理TB级的数据。

2 相关工作

2.1 决策树挖掘算法

决策树挖掘算法以其结构简单、效率高、准确度高的优点成为应用最广泛的分类算法。早期最有影响的决策树算法是

到稿日期:2013-01-27 返修日期:2013-04-13 本文受国家自然科学基金(61240046)资助。
陈湘涛(1973—),男,博士,副教授,主要研究方向为数据挖掘、并行计算,E-mail:xtchen2009@163.com;张 超(1988—),男,硕士,主要研究方向为数据挖掘、并行计算;韩 茜(1982—),女,博士生,主要研究方向为数据挖掘。

Quinlan 提出的 ID3 算法[2]，ID3 算法采用信息增益作为分裂指标，理论清晰、结构简单，但是 ID3 只能处理离散数据。随后，Quinlan 提出了能够同时处理离散数据和连续数据的 C4.5 算法[3]。

早期的决策树算法只能处理较小的数据集，为了处理较大规模数据，IBM 的研究人员在 1996 年提出了采用属性表结构的 SLIQ 算法[4]。SLIQ 算法在计算的过程中只需读取当前属性的数据而不是全部的数据，提高了算法的可扩展性。不过，SLIQ 算法的属性表结构只包含属性值和记录编号，在分裂时需要借助常驻内存的类表结构。Shafer. J 等在 SLIQ 算法基础上提出了采用全新属性表结构的 SPRINT 算法[5]。SPRINT 算法的属性表结构包含属性值、类标签和记录编号，即包含了计算属性分裂指标的所有信息，避免了类表结构，解除了算法的内存限制。

2.2 并行决策树挖掘算法

虽然 SLIQ 算法和 SPRINT 算法采用属性表结构提高了算法的可扩展性，但是随着大数据时代的到来，它们也无能为力，并行化成为了一个重要的研究方向。从数据共享和同步方法来看，并行决策树算法可分为基于共享内存的并行决策树算法、基于消息通信的并行决策树算法和基于云计算的并行决策算法 3 种[6,7]。

2.2.1 基于共享内存的并行决策树算法

共享内存系统(Shard Memory Multiprocessors System，SMP)的特点是多个处理器或计算节点共享同一物理内存，通过锁机制和间断点完成数据同步。文献[8]提出一种基于属性计数器和同步锁机的 BASIC 算法，并采用固定窗口和滑动窗口技术改进了 BASIC 算法的同步机制。文献[9]则提出了一种混合使用任务并行和数据并行策略的 SUBTREE 算法。共享内存式的挖掘算法具有效率很高、内存开销小的优点[10]；但是，它只适用于单机多核的高性能机器或者分布共享存储多处理机系统，对机器的要求高且可扩展性较差。

2.2.2 基于消息通信的并行决策树算法

消息传递是相对于进程间通信方式而言的，理论上任何支持进程间通信的并行机都支持消息传递并行程序设计，但消息传递机制在 SMP 中效率不高，消息传递机制主要应用于分布式系统。MPI[11]是消息传递领域中应用最广泛的并行编程标准，常见的并行算法通常都是基于 MPI 实现的[12,13]。

基于 MPI 的并行算法具有良好的可移植性和扩展性，适合处理计算密集型应用。由于 MPI 的设计原则是用“计算换通信”，在处理数据密集型应用时，大量数据在节点间进行交换，网络通信将成为影响系统性能的重要因素，性能会大大降低。此外，基于 MPI 的并行算法容错性较差，无法处理节点失效，如果 MPI 在运行过程出现节点失效或网络通信中断，则只有退出，所有的计算不得不重新开始。所以基于 MPI 的并行算法并不适合处理数据密集型应用。

2.2.3 基于云计算的并行决策树算法

为了处理数据密集型应用，Google 的研究人员提出了云计算[14]的概念，其核心是 MapReduce 并行模型[15]和 GFS (Google File System)[16]。MapReduce 模型包含 Map 和 Reduce 两个部分。Map 负责任务的分解，Reduce 负责收集各个任务计算的结果，并形成最终结果。GFS 则是底层的分布式文件系统。

相比 MPI，由于有分布式文件系统的支持，云计算平台的数据分布存储在各个节点。计算时各节点读取本地的数据进行处理，避免了大量数据在网络上的传输，实现了“计算向存储的迁移”，这对处理 TB 级的海量数据有很大的优势。另外，云计算平台在设计之初就假设服务器失效是常态事件，而不是意外事件，因此，云计算平台有完善的方案来处理节点失效的情况。

目前，云计算技术已经在数据挖掘领域[17-21]得到了广泛应用。基于云计算的一个非常著名的应用就是 Google 的搜索平台，在这个平台上每天都运行成千上万次的并行 CART (Classification And Regression Trees) 算法。Biswanath Panda 等提出一种基于 MapReduce 模型的并行决策树算法 PLANEN[22]。文献[23]则提出了一种采用过滤技术的并行 SPRINT 算法。

2.3 Hadoop 平台

Hadoop[24,25]是 Apache 开源组织的一个分布式计算开源框架，是 Google 云计算平台的开源实现。它的核心是 MapReduce 和 HDFS(Hadoop Distributed File System)。基于 Hadoop 的应用程序能够运行在大型集群上，并以一种可靠容错的方式并行处理 TB 级别的数据集。在国外，Hadoop 已经成为处理海量数据的事实标准。无数著名的互联网公司，如 Facebook、Yahoo、Amazon 等都在系统中使用 Hadoop。

鉴于传统的串行共享决策树挖掘算法仅适用于小数据集，无法满足海量数据挖掘的需求，而基于 Hadoop 平台的并行算法能够处理大规模的数据，本文提出一种基于 Hadoop 的共享决策树挖掘算法来满足挖掘大数据之间共享知识的需求。

3 并行共享决策树挖掘算法(PSDT)

3.1 相关定义

在文献[1]给出的共享决策树定义的基础上，给出了并行共享决策树的问题描述，以及并行环境下的分裂指标和评价标准。

定义 1(共享决策树[1]) 对于给定的数据集 D_1 和 D_2 或者给定的数据集对(D_1，D_2)，共享决策树(Shared Decision Tree，SDT)是既能对 D_1 中数据进行分类也能对 D_2 中数据分类的决策树。

一个高质量的共享决策树要满足以下要求：(1)对 D_1 和 D_2 都有良好的分类准确度；(2)具有良好的数据分布相似性，即以相似的方式划分 D_1 和 D_2。

定义 2(并行共享决策树) 对于给定的数据集 D_1 和 D_2 或者给定的数据集对(D_1，D_2)，并行共享决策树(Parallel Shared Decision Tree，PSDT)是指采用并行的策略构建的决策树，该决策树既能对 D_1 中的数据进行分类也能对 D_2 中的数据进行分类。

一个高质量的并行共享决策树除了需要满足共享决策树的要求外，还要有良好的并行性。共享决策树同时考虑了分类准确度和数据分布相似性两个因素，下面给出了并行环境下二者的定义。

定义 3(分类准确度) 假设 T 为数据集对(D_1，D_2)的并行共享决策树，则 T 的分类准确度为 $SA(T)=\min(Acc_{D_1}(T),Acc_{D_2}(T))$，其中 $Acc_{D_1}(T)$ 表示数据集 $D_i(i=1,2)$ 的分类准确度，其计算公式为：

$$Acc_{D_i}(T)=1-\frac{1}{num_{D_i}}\sum_{j=1}^{n}W_j \tag{1}$$

式中，W_j 为数据集 D_i 存储在计算节点 j 上被错误分类的元组数，num_{D_i} 为数据集 D_i 的所有元组数。

数据分布相似性描述了 D_1 和 D_2 中的数据在并行共享决策树上的分布相似性，在给出其定义之前，需要先定义类分布向量和节点分布相似性。

定义 4(类分布向量) 给定并行共享决策树 T 上的一个节点 V，数据集 $D_i(i=1,2)$在节点 V 的类分布向量(class distribution vector)为

$$CDV_i(V)=\sum_{j=1}^{n}CDV_{i,j}(V) \tag{2}$$

式中，

$$CDV_{i,j}(V)=(Cnt_{i,j}(C_1,V),Cnt_{i,j}(C_2,V),\cdots,Cnt_{i,j}(C_k,V)) \tag{3}$$

即：

$$CDV_i(V)=(\sum_{j=1}^{n}Cnt_{i,j}(C_1,V),\sum_{j=1}^{n}Cnt_{i,j}(C_2,V),\cdots,\sum_{j=1}^{n}Cnt_{i,j}(C_k,V)) \tag{4}$$

式中，n 是并行计算平台中计算节点的数量，$Cnt_{i,j}(C_k,V)$表示数据集 D_i 存储在计算节点 j 上的元组个数，这些元组属于节点且类标签为 $C_k(k=1,2,\cdots,m)$；$C_1,C_2,\cdots,C_m$ 为不同的类标签。

定义 5(节点分布相似性) 给定并行共享决策树上 T 的一个节点 V，其节点分布相似性为：

$$DNS(V)=\frac{CDV_1(V)\cdot CDV_2(V)}{\|CDV_1(V)\|\cdot\|CDV_2(V)\|} \tag{5}$$

式中，分子为向量 $CDV_1(V)$和 $CDV_2(V)$的向量积，分母为向量 $CDV_1(V)$和 $CDV_2(V)$的模的乘积。

定义 6(数据分布相似性) 并行共享决策树的数据分布相似性定义为：

$$DS(T)=\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{n}\sum_{j=1}^{n}DSN(V_{i,j})) \tag{6}$$

式中，V_{ij} 表示存储在计算节点 i 的非根节点，m 为集群中计算节点数量，n 为 T 中非根节点数量。

为了使并行共享决策树同时具有良好的数据分布相似性和分类准确度，并行共享决策树的质量评价标准和分裂指标定义如下：

定义 7(质量评价标准) 并行共享决策树的质量评价标准为：

$$SDTQ(T)=\min(SA(T),DS(T))*avg(SA(T),DS(T)) \tag{7}$$

SDTQ 同时考虑了 SA 和 DS 两个因素，并且突出了 SA 和 DS 中较小元素的作用。SDTQ 能够保证一个 SDTQ 得分高的共享决策树一定是高质量的共享决策树。

定义 8(分裂指标) 假设 V 为并行共享决策树 T 的一个待分裂节点，A 是节点 V 的一个待分裂属性，a_v 为属性 A 中的一个可能的分裂点，则 T 的分裂指标为：

$$ID(A,a_v)=\omega_{IG}*IG(A,a_v)+\omega_{DS}*DSN(A,a_v) \tag{8}$$

式中，$ID(A,a_v)$为信息增益，其计算与 ID3 相同；$DSN(A,a_v)$是 A 和 a_v 将节点划分后的两个子节点的 DSN 值的平均值；ω_{IG}和 ω_{DS}分别为信息增益和数据分布相似性的权重。

3.2 PSDT 算法的并行思想

决策树的生成过程是一个迭代过程。在这个过程中存在以下并行性。

属性并行性：在节点分裂时，需要计算每个属性的分裂指标，而计算分裂指标任务只涉及当前属性的数据，与其他属性的数据无关，即同一节点内部的各个属性是独立的。因此，可以并行地计算分裂指标，即属性并行性。

节点并行性：决策树中的每一个节点都是一个独立的数据集，其分裂任务与其兄弟节点的分裂任务是互不相干的，即同一层的节点之间是独立的。因此，同一层的节点可以同时进行分裂任务，即节点并行性。

排序并行性：由属性并行性和节点并行性可知，决策树同一层的所有节点的所有属性之间都是独立的，可以并行地对这些属性进行预排序[5]，即排序并行性。

虽然共享决策树的目的是挖掘共享知识，但它仍然是一种决策树，同样具有决策树的性质。因此，决策树的并行策略[26,27]同样适用共享决策树。然而，如何将属性并行、节点并行、排序并行与 MapReduce 模型紧密结合则是并行共享决策算法面临的主要问题。

3.2.1 属性并行

在决策树分裂的时候，为了寻找最佳分裂点，需要计算每一个属性的分裂指标。当训练数据有成千上万个属性时，并行地计算这些属性是提高算法性能的重要途径之一。采用属性表结构[5]很容易实现属性并行，将数据集拆分成属性表后就可以并行处理这些属性表。

MapReduce 的 Map 是作为映射功能而存在的。通过定义分发规则，可以将不同的〈key，value〉映射到不同的Reducer进行处理。那么，利用 MapReduce 中 Map 的映射功能，在属性表上附上属性 ID，定义分发规则：具有相同属性 ID 的属性表分发到相同的 Reducer，同一节点的属性表就可以被同时处理，然后映射到不同的 Reducer 执行相应的计算分裂指标任务。

3.2.2 节点并行

在构造决策树的过程中，串行算法对同一层的节点逐个进行递归分裂。而事实上决策树中同一层的节点之间是相互独立的，可以并行处理。利用 MapReduce 中 Map 的映射功能，在属性表上附上节点 ID，通过定义分发规则：具有相同节点 ID 的数据分发到相同的 Reducer，同一层的所有节点就可以被同时处理，然后映射到不同的 Reducer 执行相应的分裂。

显然，在属性表上附上相应的节点 ID 和属性 ID，定义 Map 的分发规则如下：具有相同节点 ID 和属性 ID 的属性表分发到同一个 Reducer，即可同时实现属性并行和节点并行，如图 1 所示。

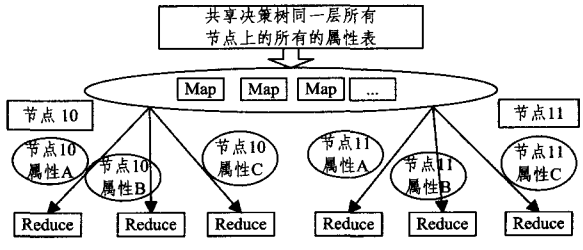

图 1 属性并行和节点并行

3.2.3 排序并行

在计算分裂指标值时经常需要排序。当训练数据较大时，排序操作会严重影响算法的时间性能。利用 MapReduce 天然的排序特征[24,25]，可以很容易解决这个问题。

在 MapReduce 执行过程中的 Map 阶段，被分发到同一

个 Reducer 的〈key,value〉键值对被收集起来后会根据 key 进行排序;在 Reduce 阶段,来自不同 Map 端的〈key,value〉键值对被收集起来后也会根据 key 进行排序。即同一个 Map 的输出是有序的,同一个 Reducer 的输入也是有序的。基于 MapReduce 的排序特征,将属性表的属性值放入 key 中,定义 key 的排序规则:根据属性值进行排序,即可实现对属性表的并行排序。

综合上述的 3 种并行性,可以设计属性表的结构如图 2 所示。

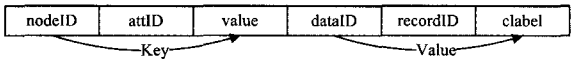

图 2 属性表结构

其中,nodeID 为节点编号,attID 为属性编号,value 为属性值,dataID 为数据集编号,recordID 为记录编号,clabel 为类标签。

3.3 PSDT 算法

PSDT 算法的流程如图 3 所示。每次分裂时,PSDT 算法都利用 2 个 Job 来完成,其中 Job1 负责根据式(8)并行地计算共享树当前层每一个节点每一个属性的分裂指标值;Job2 负责并行地对共享树当前层的每一个节点执行分裂。

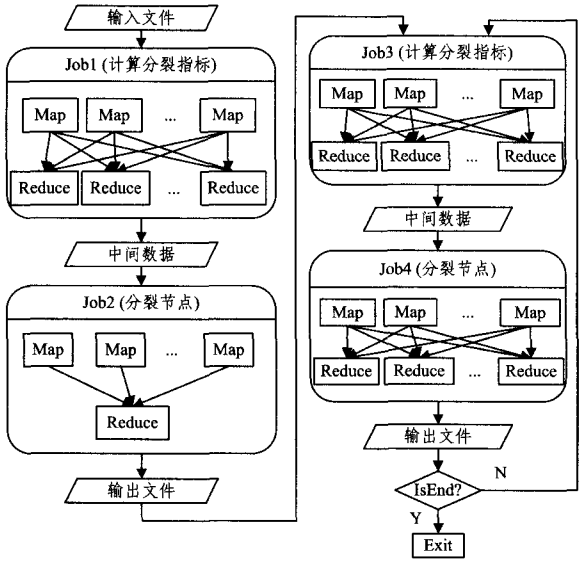

图 3 PSDT 算法流程

其中 Job1 和 Job2 进行根节点分裂,Job3、Job4 表示循环分裂;Job1 的输出为中间数据,Job2 的输出文件为节点分裂结果;由于第一次分裂时只有根节点一个待分裂节点,Job2 将 reduce 数量强制设为 1。

3.3.1 计算分裂指标(Job1)

根据 3.2 节中的分析,Job1 的算法设计如 Algorithm 1 所示。为了实现属性并行和节点并行,Job1 的 Map 根据节点编号和属性编号分发数据;同时为了排序并行,Job1 采用多次排序技术[25]借用 MapReduce 本身的排序实现属性表的预排序。

由于每个 Reducer 可能需要处理多个属性的数据,在 Reduce 阶段需要将数据进行分组,然后才能调用 reduce 函数计算分裂指标值。Job1reduce 的分组规则与 Map 端的分发规则相同。为了减少数据冗余,reduce 函数没有将分裂信息附在属性表结构上[17],而是单独输出一条属性分裂信息(格式为〈〈nodeID, flog〉,〈recordID, attID, ID, Split〉〉,其中 flog 为分裂信息标志位,用以区别于属性表;ID 为分裂指标值,Split 为分裂点)。同时为了便于在 Job2 进行预排序(见 3.3.2 节),Job1 输出时设置属性表为〈〈nodeID, dataID〉,〈recordID, attID, value, calbel〉〉。

Algorithm 1 Job1

```
Map 阶段:
分发规则:nodeID、attID 相同的属性表发送到同一 Reducer
排序规则:在不改变 nodeID 和 attID 排序结果的前提下,根据 value 对属性表进行排序
Input:待分裂的属性表,格式如图 3 所示
Output:属性表,格式如图 3 所示
Method:
  Partition attribute list// 分发属性表
Reduce 阶段:
分组规则:nodeID 和 attID 相同的属性表划分到同一组
Input:Map 阶段的输出
Output:属性表和分裂指标信息
Method:
  For each group do
    Initialize histogram//初始化类直方图
    Compute ID according to ID()//根据式(8)计算分裂指标
  End for
```

3.3.2 执行分裂(Job2)

Job2 负责执行节点分裂,其具体任务是根据 Job1 的计算结果选择最佳分裂点,然后构造 hash 表,并根据 hash 表分裂节点。Job2 的 Map 根据 nodeID 分发数据;Job2 的 Reduce 负责具体分裂,分裂过程如 Algorithm 2 所示。此外,Job2 采用多次排序技术对 Reduce 的输入数据进行了预排序,确保先遍历分裂信息后遍历待分裂数据,即一次遍历完成寻找最佳分裂点和构造 hash 表两个任务,减少一次数据遍历。

Algorithm 2 Job2Reduce

```
分组规则:nodeID 相同的属性表划分到同一组
Input:Job2 Map 的输出
Output:属性表,格式如图 3 所示
Method:
  For each group do
    If current node is leaf node//当前节点为叶节点
      Return the class label//返回叶节点的类标签
    Else
      Find the best split
      Create hash table
      For each 〈key,value〉 pair do
        If the 〈key,value〉 pair ∈left child node
          Assign the 〈key,value〉 pair to the left child node //划分到左子结点
        else
        Assign the 〈key,value〉pair to the right child node //划分到右子节点
        End if
    End for
  End if
End for
```

3.3.3 循环分裂

传统的串行共享决策树算法采用递归的构造方式，移植到 Hadoop 平台后使用了循环的方式。在 MapReduce 模型中，Job 的输出需要写入文件，不同 Job 之间通过文件进行沟通。因此，如果要在 MapReduce 模型中实现循环就必须设计好输入输出文件夹在 HDFS 中的命名，以便实现循环分裂。例如，将分裂过程中 Job1 的输出文件命名为 Middle1，Middle2，…，Job2 的输出文件命名为 level1，level2，…。Job1 的输出作为 Job2 的输入，Job2 的输出作为下次分裂过程中 Job1 的输入，当没有待分裂数据时，算法终止分裂。

4 混合并行共享决策树挖掘算法

4.1 PSDT 算法存在的问题

PSDT 算法采用并行决策树算法常用的属性表结构来实现并行化，但是属性表结构增加了存储代价和分裂复杂度。表 1 显示了数据记录结构的训练数据拆分成属性表后的存储代价。采用 MapReduce 模型的决策树算法在 Map 与 Reduce 之间、Job 之间都需要进行 I/O，属性表的存储代价会更加高昂。

表 1 数据记录与属性表结构存储大小

数据记录(M)	属性表(M)
7.6	25.9
38.1	136.8
76.3	275.5
114.5	414.3
229.1	830.4
458.4	1638.4
917	3276.8

在执行节点分裂时，基于属性表的决策树算法需要借助与训练集大小成正比的 hash 表。随着训练集的增长，这部分代价变得越来越昂贵。这个过程对基于 Hadoop 的决策树算法的影响也更加严重。由于需要先构造 hash 表，才能执行分裂，不可避免地需要多次遍历数据，而对大数据集的遍历会严重影响算法的时间性能。

4.2 HPSDT 算法思想

针对 PSDT 算法存在的问题，考虑到数据记录结构虽然不适合并行计算分裂指标，但其存储代价要比属性表结构小很多；而且分裂过程[2,3]简单，不需要构造 hash 表，本文提出一种基于 Hadoop 的混合并行共享决策树挖掘算法（Hybrid Parallel Shared Decision Tree，HPSDT）。HPSDT 算法混合使用属性表和数据记录结构，在计算分裂指标阶段采用属性表结构，在分裂阶段采用数据记录结构，既利用属性表进行并行计算，又简化了分裂过程，大大减少了 I/O 量。

4.3 HPSDT 算法

HPSDT 算法与 PSDT 算法的处理流程基本一致，都是采用两个 Job 完成分裂，Job1 负责计算分裂指标值，Job2 负责执行分裂。二者的不同之处有两点，一是 HPSDT 在执行分裂时采用数据记录结构简化了分裂过程；二是 HPSDT 在两个 Job 的输入输出用数据记录取代属性表结构，减少了I/O。

HPSDT 算法的分裂过程比较简单，如 Algorithm 3 所示。HPSDT 算法在执行分裂时根据最佳分裂点就可以直接进行分裂，不需要构造 hash 表。同时 HPSDT 算法采用多次技术确保在执行分裂时先遍历所有分裂信息，找到最佳分裂点，然后分裂数据，即一次遍历完成整个分裂过程。

Algorithm 3 HPSDT Job2Reduce

```
分组规则：nodeID 相同的属性表划分到同一组
Input：Job2 Map 的输出
Output：数据记录，格式为〈〈nodeID, dataID〉, Text〉，其中 Text =
        "ID, value1, value2, …, clabel"
Method：
  For each group do
    If current node is leaf node//当前节点为叶节点
      Return the class label//返回叶节点的类标签
    Else
      Find the best split
      For each record  do
        If the value of split attribute < best split
              //该记录的分裂属性值<最佳分裂点
          Assign the record to the left child node
        else
          Assign the record to the right child node
          End if
        End for
      End if
  End for
```

在 HPSDT 算法中，属性表结构只出现在 Job1 的 Map 与 Reduce 之间，而其他阶段都采用数据记录。表 2 显示了 PSDT 与 HPSDT 算法在一次分裂过程中各个阶段的 I/O 对比（根节点除外）。假设输入的数据记录大小为 114.5M，由表 1 可知对应的属性表大小为 414.3M。

表 2 PSDT 与 HPSDT 在一次分裂过程中的 I/O 对比

分裂阶段	HPSDT(M)	PSDT(M)	比率
Job1Map	114.5+414.3	414.3+414.3	0.64
Job1Reduce	414.3+0.17	414.3+414.47	0.5
Job2Map	114.5+114.5	414.3+414.3	0.28
Job2Reduce	114.5+114.5	828.6+828.6	0.14
Total	1401.27	4143.17	0.34

注："+"前后分别为输入数据大小和输出数据大小

表 2 中 HPSDT 的 Job1Reduce 阶段的输出仅为 0.17M 是因为在它仅仅输出了属性的分裂信息（实验数据，详见 5.1 节，共 5114 个属性，每个属性的分裂信息大小约为0.035kB），没有输出数据。由于 Job1 的输入输出和 Job2 的输入类型一致，HPSDT 在 Job2 的输入阶段采用多路径输入技术[24]，即 Job2 的输入包括 Job1 的输入和输出，其中 Job1 的输入为待分裂数据，Job1 的输出为分裂指标信息。因此，Job1 只需要输出分裂信息。PSDT 算法的 Job2Reduce 阶段由于需要遍历数据两次，因此产生了两次读写操作.

由表 2 可知，在一次节点分裂过程中，HPSDT 的 I/O 量仅为 PSDT 的 0.34，I/O 量大大减少。共享决策树的构造过程是一个循环进行节点分裂的过程，因此，在整个共享决策树的构造过程中，HPSDT 的 I/O 量同样仅为 PSDT 的 0.34。

需要说明的是，HPSDT 算法为此付出了 CPU 代价。由于 Job2 的输出为数据记录结构，每次分裂时都需要重新生成属性表，消耗了 CPU 资源。而 PSDT 算法除了根节点外，节点的分裂不需要重新生成属性表。不过，对基于 MapReduce 的并行算法而言这是完全可以接受的，因为在云计算平台的

CPU、内存、I/O 3个要素中，I/O通常是最重要的资源。通常计算节点的CPU利用率都远低于100%，而I/O则通常是瓶颈。因此，对于基于MapReduce的并行算法，CPU换I/O的策略是可行的。

5 实验结果

5.1 实验数据及实验环境

实验数据来自肯特岗生物医学数据集中的乳腺癌数据集(Brease Canncer，BC)和中枢神经系统数据集(Central Nervous System*，CN)。BC和CN数据集共有5114个共享属性，本文选择这些共享属性数据作为实验数据，并按照文献[1]中的方式合成实验数据。

实验平台是一个由刀片服务器搭建的集群。每台刀片服务器的配置均为4颗Intel i3 CPU、2.93GHZ，3.2GB内存，1T磁盘。每台服务器均安装了Hadoop-0.20软件和Javal.7。

5.2 并行性实验

为了验证PSDT算法和HPSDT算法的并行策略的有效性，本文将它们与串行的SDT算法进行对比测试。同时，为了验证混合数据结构的效果，本文详细对比了PSDT算法和HPSDT算法。PSDT算法和HPSDT算法都运行在3台刀片服务器搭建的Hadoop集群上，SDT算法则运行于同配置的单台刀片服务器，其结果如图4、图5所示。

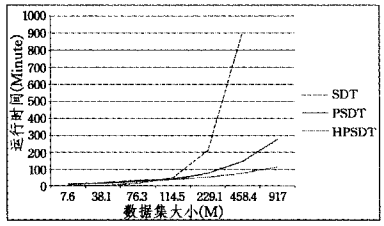

图4 SDT、PSDT、HPSDT的运行时间

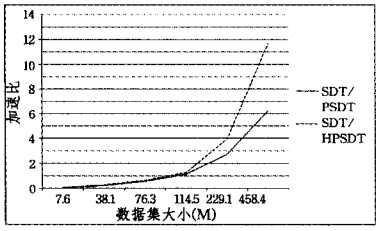

图5 PSDT、HPSDT相对SDT的加速比

从图4中可以看出，数据集的大小为917M时，串行的SDT算法无法运行，而PSDT和HPSDT算法都能继续运行，即并行算法拥有良好的可扩展性。图5则说明在数据集较小的时候，SDT算法比PSDT和HPSDT算法都要快(加速比小于1)；随着数据集增大，PSDT和HPSDT算法的并行性越来越好。实验数据表明在数据集的大小为458.4M时，PSDT和HPSDT的加速比分别达到了6.24和11.65。

表3详细对比了PSD和HPSDT算法的时间性能，从中可知，HPSDT算法优于PSDT算法。即使数据集较小时，PSDT与HPSDT的运行时间比率仍大于1.1；而且随着数据集的增大，PSDT算法采用属性表结构带来的I/O问题越来越严重，HPSDT算法的优越性越来越明显，当数据集的大小为917M时，二者的运行时间比率达到了2.45。

表3 PSDT与HPSDT的运行时间对比

数据集大小(M)	PSDT(minute)	HPSDT(minute)	运行时间比率
7.6	12.95	11.7	1.11
38.1	20.98	18.76	1.12
76.3	34.55	30.35	1.14
114.5	43.55	39.65	1.15
229.1	76.83	52.93	1.45
458.4	144.7	77.5	1.87
917	278.33	113.73	2.45

5.3 集群规模实验

通过在不同节点数量的集群上运行HPSDT算法来观察Hadoop集群规模对算法的影响，实验结果如图6所示。

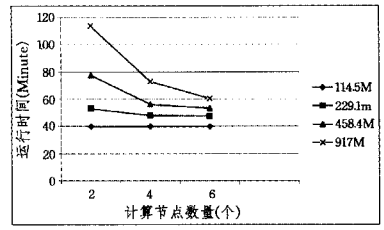

图6 集群规模实验

由图6可知，随着计算节点数量的不断增加，HPSDT算法的运行时间逐步减少；而且随着数据集的增大，这种趋势更加明显。值得注意的是在数据集较小时，计算节点的增加并没有带来明显的性能提升。这是因为小数据集对资源要求低，少量的计算节点就能实现较好的并行性，更多的节点带来的加速效果有限，反而带来更多的通信代价。

结束语 (1) 现有串行共享决策树挖掘算法受到运行平台的内存、计算能力和I/O等因素限制，无法满足挖掘大数据集的需求。为此，提出了一种基于云计算的并行共享决策树挖掘算法PSDT。该算法通过引入并行决策树的思想和云计算平台，能够在大规模的计算集群上处理TB级的海量数据。

(2) PSDT算法具有良好的并行性和扩展性。在数据集的大小为458.1M时，其加速比达到了6.24。相比串行的共享决策树挖掘算法，PSDT能够处理更大规模的数据。

(3) 针对PSDT算法采用属性表结构的弊端，进一步提出一种混合并行共享决策树挖掘算法HPSDT。该算法综合了属性表结构和数据记录结构的优点，大大减少了I/O操作。实验结果表明，HPSDT具有更好的性能，在数据集大小为917M时，PSDT与HPSDT的运行时间比率即可达到2.45；而且随着数据集的增大，HPSDT的优越性更加明显。

参考文献

[1] Dong Guo-zhu, Han Qian. Mining Shared Decision Trees between Datasets[R]. OH. USA: Wright State University/OhioLINK, 2010

[2] Quinlan J R. Induction of decision trees [J]. Machine Learning, 1986(1): 81-106

[3] Quinlan J R. C4.5: Programs for machine learning [M]. San Francisco: Morgan Kaufmann Publishers Inc., 1992

[4] Manish M, Rakesh A, Jorma R. SLIQ: A fast scalable classier for data mining[C] // EDBT' 96. London: Springer-Verlag, 1996: 18-32

[5] John C S, Rakesh A, Manish M. SPRINT: A scalable parallel classier for data mining[C]//22th VLDB Conference. San Fran-

cisco:Morgan Kaufmann Publishers Inc. ,1996:544-555
[6] Hayes B. Cloud computing [J]. Communications of the ACM, 2008,51(7):9-11
[7] Armbrust M,Stoica I,Zaharia M,et al. A view of cloud computing [J]. Communications of the ACM,2010,53(4):50-58
[8] Mohammed J Z,Ho Ching-tien,Rakesh A. Parallel classification for data mining on shared-memory multiprocessors [C]// ICDE: IEEE International Conference on Data Engineering. Sydney: IEEE Computer Society,1999:198-205
[9] Mohammed J Z. Parallel and distributed association mining: a survey [J]. IEEE Concurrency,1999,7(4):14-25
[10] Ruo-ming J,Ge Yang,Gagan A. Shared memory parallelizetion of data mining algorithms: techniques, programming interface, and performance[J]. Transactions on Knowledge and Data Engineering,2005,17(1):71-89
[11] Peter S P. Parallel Programming with MPI [M]. San Francisco: Morgan Kaufmann Publishers Inc. ,1997
[12] Gropp W,Lusk E,Skjellum A. Using MPI portable parallel programming with the message-passing interface [M]. Cambridge: MIT Press,1999
[13] Nuno A,João G,Fernando S. Exploiting Parallelism in Decision Tree Induction[C]// ECML/PKDD Workshop on Parallel and Distributed computing for Machine Learning. 2003:13-22
[14] Wu Rong,Liang Shuai,Liao Hua-ming. Cyclic workflow execution mechanism on top of MapReduce framework[C]// Seventh International Conference on Semantics, Knowledge and Grids. Washington,DC: IEEE Computer Society,2011:28-35
[15] Jeffrey D,Sanjay G. MapReduce: Simplified Data Processing on Large Clusters [J]. Communications of the ACM,2008,51(1): 107-113
[16] Sanjay G,Howard G,Leung S T. The Google file system [C]// the Nineteenth ACM Symposium on Operating Systems Principles(SOSP' 03). Bolton Landing,NY,USA,2003,37(5):29-43
[17] Kang U,Tsourakakis C,Appel A P,et al. HADI: fast diameter estimation and mining in massive graphs with Hadoop[R]. Pittsburgh:Carnegie Mellon University,2008
[18] Lin J, Dyer C. Data-intensive text processing with MapReduce [M]. Morgan and Claypool Publishers,2010
[19] Ene A,Im S,Moseley B. Fast clustering using MapReduce[C]// Proceeding of the 17th ACM SIGKDD Inernational Conference on Knowledge Discovery and Data Ming. New York: ACM, 2011:681-689
[20] 向小军,高阳,商琳,等. 基于 Hadoop 平台的海量文本分类的并行化[J]. 计算机科学,2011,38(10):184-188
[21] 赵卫中,马慧芳,傅燕翔,等. 基于云计算平台 Hadoop 的并行 k-means 聚类算法设计研究[J]. 计算机科学,2011,38(10):166-168
[22] Biswanath P,Joshua S H,Sugato B,et al. PLANET: Massi-vely parallel learning of tree ensembles with MapReduce[C]// 35th International Conference on Very Large Data Bases (VLDB-2009). 2009:1426-1437
[23] Lu Qiu,Cheng Xiao-hui. The Research of Decision Tree Mining Based on Hadoop[C]// 9th International Conference on Fuzzy Systems and Knowledge Discovery(FSKD 2012). 2012:798-801
[24] Apache. Hadoop[EB/OL]. http://hadoop. apache. org/,2012-10-20
[25] White T. Hadoop:权威指南 [M]. 曾大聃,周傲英,译. 北京:清华大学出版社,2010
[26] Mohammed J Z. Parallel and Distributed Data Mining: An Introduction[C]// Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems. London,UK: SIGKDD,2000:1-23
[27] Ruoming J, Gagan A. Communication and memory efficient parallel decision tree construction[C]// the third SIAM International Conference on Data Mining. San Fra ncisco: Society for Industrial & Applied,2003:119-129

(上接第 186 页)

[12] Niemelä I,Simons P,Syrjänen T. Smodels: a system for answer set programming [C] // Proceedings of the 8th International Workshop on Non-Monotonic Reasoning. Brenkenridge, Colorado,USA,2000
[13] Lin Fang-zhen,Zhao Yu-ting. ASSAT: computing answer sets of a logic program by SAT slover[J]. Artificial Intelligence,2004, 157(1/2):115-137
[14] Lifschitz V, Turner H. Splitting a logic program [C]// Proceedings of the Eleventh International Conference on Logic Programming. 2008:23-37
[15] Baral C. Knowledge Representation, Reasoning, and Declarative Problem Solving [M]. Cambridge Press,2003
[16] Heljanko K, Niemelä I. Bounded LTL model checking with stable models [J]. Theory and Practice of Logic Programming, 2003,4(3):519-550
[17] Leone N,Pfeifer G,Faber W,et al. The DLV system for knowledge representation and reasoning [J]. ACM Transactions on Conputational Logic,2006,3(7):499-562
[18] 陆汝钤. 计算机语言的形式语义[M]. 北京:科学出版社,1992
[19] Brookes S D,Hoare C A R,Roscoe A W. A theory of communicating sequential processes [J]. Journal of the ACM, 1984, 31 (3):560-599
[20] Palikareva H, Ouaknine J, Roscoe A W. SAT-solving in CSP trace refinement [J]. Science of Computer Programming. Special issue on Automated Verification of Critical Systems, 2012, 77 (10/11):1178-1197
[21] Ranzato F. On the completeness of model checking[C]// Proc. ESOP'01,LCNS 2028. Springer,1999:137-154
[22] Giacobazzi R, Ranzato F. States vs. traces in model checking [C]// Proc. 9th International Static Analysis Symposium (SAS' 02). LNCS 2477,2002:461-476
[23] Howard Y,Gruner S,Gravell A,et al. Model-based trace checking [C]// SoftTest: UK Software Testing Research Workshop II. 2003