doi: 10.7690/bgzdh.2021.12.001

基于单位层次树的数据归并算法

罗晓玲，陈财森，向阳霞，金传洋
（陆军装甲兵学院信息通信系，北京 100072）

摘要：为提高数据挖掘处理效率，提出一种基于单位层次树的归并计算方法。以装备维修保障相关数据为例，建立叶子节点的自下而上逐层归并计算模型，对 MapReduce 并行计算模型进行改进，采用完全分布式模式实现 Hadoop Map/Reduce 分布式处理架构，实现面向单位层次树的属性约简与数据归并，对 MapReduce 分布式模型的加速比和可扩展性进行分析。结果表明：该方法取得了较好的效果，具有一定理论价值。

关键词：数据预处理；归并计算；分布式计算
中图分类号：TJ07　　**文献标志码：**A

Data Merge Algorithm Based on Unit Hierarchy Tree

Luo Xiaoling, Chen Caisen, Xiang Yangxia, Jin Chuanyang
(*Department of Information & Communication*, *Army Academy of Armored Forces*, *Beijing* 100072, *China*)

Abstract: In order to improve the efficiency of data mining, a merge calculation algorithm based on unit hierarchical tree is proposed. Taking equipment maintenance support related data as example, establish a bottom-up layer-by-layer merge calculation model of leaf nodes, improve the MapReduce parallel calculation model, and adopt a fully distributed mode to implement the Hadoop Map/Reduce distributed processing architecture to achieve unit-oriented level. The attribute reduction of the tree and data merging are used to analyze the speedup and scalability of the MapReduce distributed model. The results show that the method has achieved good results and has certain theoretical value.

Keywords: data preprocessing; merging algorithm; distributed computing

0　引言

为了从大量杂乱无章、难以理解的海量装备维修数据中抽取并推导出有价值的信息，需对其进行特征选取与预处理，以求有效简化计算难度，提高效率。

在数据预处理中，数据归约是指在对挖掘任务和数据本身内容理解的基础上，在保持原始数据完整性的前提下，发现数据的有用特征，删除冗余知识，最大限度地精简数据量、缩减数据规模，实现更高的数据挖掘处理精度，提高挖掘效率。装备维修数据具有明显的单位层次结构，如连、营、团、旅。而采集的装备修理信息，均属于本级单位层次，若想查看某团级的所有装备维修信息，则需逐层计算。为了能够直接从相应层次获取所有下级层次的数据，笔者提出基于单位层次树的归并计算方法，进一步对装备维修数据进行属性约简和数据重构[1-2]。

1　基于单位层次树的归并模型

对于具有单位层次结构的数据，层次树能够很好地展现单位及其信息的逻辑关系。由于数据信息繁多，展示成千上万个节点的树结构杂乱且没有太大意义；因此，需要对数据信息进行归并统计计算，把冗余属性剔除掉，将同类型关键数据合并，减少或避免结果数据的复杂度和不一致性，一定程度上达到数据清洗的作用。

依据单位编码能够构建单位层次树，把单位相关信息落在单位层次树的相应节点上。若要对层次树中某一层级的单位信息进行查询、检索，意味着要对该单位的所有子树进行统计计算，在实际应用中计算量大且耗时长。为了能够快速获取层次树中某单位的信息，对树中数据进行预处理时，从叶子节点开始自下而上进行归并，完成对单位层次树中数据信息的规整，将统计信息放在每层相应的非叶子节点上，得到一棵归并树。当需要快速查询、检索某一层级单位的统计信息时，能够直接从相应的节点获取，不需要再逐层计算，节省了时间并提升了效率。

1.1　归并树的形成

图 1 的单位层次树就是一棵归并树，树中叶子

收稿日期：2021-08-15；**修回日期：**2021-09-24
作者简介：罗晓玲(1985—)，女，福建人，硕士，讲师，从事计算机基础教学、人工智能、陆上无人平台研究。E-mail: 26073247@qq.com。

节点(图中白色节点)上记录着单位相关信息，非叶子节点(图中黑色节点)上仅储存对其子树进行归并计算后得到的统计信息。如：节点 L 上储存着节点 Q、节点 R 归并后的统计信息；节点 E 上储存着节点 K、节点 L 归并后的统计信息。当检索节点 E 所指单位的信息时，即可快速查询到以 E 为根节点的子树中所有单位信息归并后的统计信息。

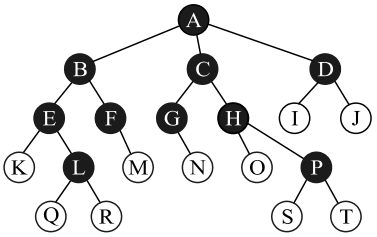

图 1 归并树

原始单位层次树的结构在实际应用中可能存在 2 种情况：第 1 种情况，单位层次树中仅叶子节点上存有单位相关信息，非叶子节点上未记录单位相关信息(这是一棵理想化的单位层次树)，此时能够直接对单位层次树进行自下而上的归并计算，将统计信息储存在相应非叶子节点上，形成归并树；第 2 种情况，除了叶子节点上存有单位相关信息外，非叶子节点上也存有单位相关信息，图 2 描述了部门工作人数的单位层次树(员工人数=下级部门员工人数+本部门员工人数)，对于节点 1.2，其员工人数为节点 1.2.1、节点 1.2.2 加上节点 1.2 自身员工人数的总和。这种情况是普遍存在的。

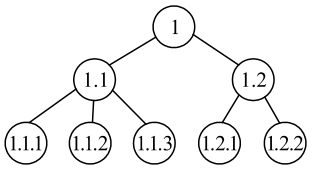

图 2 单位层次树

对于第 2 种情况，为了简化归并、保证计算模型的一致性，需要对单位层次树进行同构化处理：对存有单位相关信息的非叶子节点虚拟一个孩子节点，如图 3 中“本级”节点。

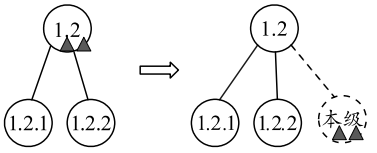

图 3 单位层次树同构化

将非叶子节点上(如节点 1.2)的单位相关信息记录在虚拟的孩子节点上(如“本级”节点)。如此一来，虚拟孩子节点与该非叶子节点的其他孩子节点(如节点 1.2.1 与节点 1.2.2)是兄弟节点，非叶子节点上记录其子节点(如节点 1.2.1、节点 1.2.2 及“本级”节点)归并后的信息。此时将第 2 种情况的单位层次树结构改造为第 1 种情况，能够对改造后的单位层次树进行归并计算，得到归并树。

1.2 归并计算模型

从原始的单位层次树到归并计算后的单位层次树，需建立从叶子节点开始自下而上逐层归并计算模型。笔者建立的归并模型简记关系模式为

$$R(U)\text{或 }R(A_1, A_2,\cdots, A_n)。$$

其中：R 为关系名；U 为组成该关系的属性集合；$A_1, A_2,\cdots, A_n$ 为数据属性；$A_i(i=1, 2,\cdots, n)$为有限集，其基数为 $m_i(i=1, 2,\cdots, n)$。给定一组域 $D_1, D_2,\cdots, D_n$(这些域可以是相同的)，为属性组 U 中属性所来自的域，即可知不同的属性 A_i 可以来自同一个域。

对节点的属性分别进行归并计算：

$$F(U)=(f_1(A_1), f_2(A_2),\cdots, f_n(A_n))。\tag{1}$$

其中：$f_i(A_i)\in\Theta$，$\Theta=\{\sum, -, \max, \min, \cup, \text{replace}, \text{count}\cdots\}$为归并算子的集合，不同属性的值可能采用不同的归并算子进行计算。归并完成后得到 $U(A_1, A_2,\cdots, A_n)$对应的统计结果 $D(d_1, d_2,\cdots, d_n)$，其中 d_i 为节点属性 A_i 经过归并计算后的统计值，将 $D(d_1, d_2,\cdots, d_n)$储存在非叶子节点。

2 基于单位层次树的增量模型

2.1 数据增量的处理

在实际情况中，考虑到数据量的状态，将其分为“静态数据”与“动态数据”：“静态数据”为某事务或任务完结后，数据量固定，不再发生变化；“动态数据”为事务信息或数据随着时间的推移持续更新，数据量不断增长。对于“静态数据”，预先计算统计结果并储存，然后建立索引，通过检索直接获得结果；对于“动态数据”，任何一次的计算统计值都会发生变化，若采用以某一时间点为结点进行计算的方式，在下一个时间结点计算时会把前一个时间结点的数据集重新计算一遍，计算效率低且负担重。

笔者提出一种方法：以某一时间或状态为结点，预先对数据集进行计算统计，再对该结点之后产生的数据增量进行计算，将最后 2 个计算结果叠加融合。该方法把某阶段的统计信息存储下来，在闲时做大量的计算处理，之后只是对增量进行归并计算，而增量数据只是整个数据集中很小的子集，计算增量数据后将结果叠加合并，避免对数据集重复计算，处理速度更快，效率更高。

增量数据计算步骤：

1) 设 T_0=0，以 T_1 为时间结点，对 T_0 至 T_1 时间段产生的数据集 S_1 进行统计计算并储存结果 R_1；

2) 以 T_2 为另一时间结点(T_2＞T_1)，对 T_1 至 T_2 时间段产生的数据增量 S_2 进行统计计算并储存结果 R_2；

3) 将结果 R_1 与 R_2 进行叠加融合得到 R'；

4) 令 T_1=T_2，S_1=S_1+S_2，R_1=R'；若 S_2=0，算法结束；否则重复步骤 2)。

2.2　增量叠加模型

在计算结果叠加过程中，根据数据属性的不同，叠加融合方式也有所不同。比如：属性 A_1 表示数据的最大值，此时只需要比较 S_1、S_2 数据集的最大值，若 max(S_2)＞max(S_1)，则进行替代运算。属性 A_2 表示数据的平均值，若 S_1 的数据量为 n_1，均值为 μ_1；增量 S_2 的数据量为 n_2，增量均值为 μ_2；则 S_1+S_2 的均值为$(n_1\mu_1+n_2\mu_2)/(n_1+n_2)$。叠加的计算方式根据特定的算子来集成，算子不同，需要的辅助信息不一样。

笔者建立的增量结果叠加模型如下：

已知原数据集非叶子节点属性 $U(A_1, A_2,\cdots, A_n)$ 的归并统计结果为 $D(d_1, d_2,\cdots, d_n)$，增量数据集归并统计结果 $D'(d_1', d_2',\cdots, d_n')$，根据节点属性 $A_1, A_2,\cdots, A_n$ 的归并算子分别进行叠加计算：

$$\Phi(D,\ D')=(\varphi_1(d_1,d_1'),\varphi_2(d_2,d_2'),\cdots,\varphi_n(d_n,d_n'))。\quad (2)$$

其中 $\varphi_i(d_i,d_i')\in\Omega$，$\Omega$ 为叠加算子的集合：

$$\Omega=\{\Sigma,-,\max,\min,(n_1\mu_1+n_2\mu_2)/(n_1+n_2)\cdots\}。$$

不同属性的值可能采用不同的归并算子进行计算，因此对统计结果进行叠加时采用不同的叠加算子进行合并。

3　基于 MapReduce 的归并模型改进

按层级进行数据归并是上级装备管理机关进行统计分析和计划编配的基础操作。由于全军单位的组织结构树较为庞大，装备业务涉及范围宽、装备种类多、装备数量大；因此，提高归并处理的效率颇为重要。

在实践中，通常会采取多线程的方式进行归并处理，比如在全部数据单位层次树的第二或第三层的单位结点上开单独线程进行归并，确实能够较为有效地提高处理效率；但多线程处理也受一些因素的限制，比如操作系统对线程的调度和管理能力、内存资源，更重要的是 CPU 的内核资源数量。当线程数超过内核数的 3 倍以后，归并计算一旦开始，系统负载快速达到 90%以上，计算机基本上需要等到整个归并计算处理完才会响应其他操作，计算速度慢、效率低。

单机处理难以突破瓶颈，可以用多机并行处理的方式加以解决，通过指定不同的机器处理不同二级或三级单位数据的归并处理，则需要进行人工配置与管理。MapReduce 分布式/并行模型提供了很好的解决思路，可通过较大规模的计算机集群，实现 TB 级海量数据并行的可靠容错处理；但传统的 MapReduce 模型对关联性强的数据处理效率低[3]，对应树型结构数据，在 Reduce 阶段可以用多关键字排序的方式代替单关键字排序。笔者提出一种改进，以组合关键字-值对<(key1, key2,…, keyn), value>的方式代替传统的单键值对<key, value>，即在逻辑上可以通过 Keys(key1, key2,…, keyn)的运算处理，如 hash 或键值串拼接等，将组合关键字映射为单关键字。

3.1　面向树型结构的 Mapper/Chain Reducer 模型

传统 MapReduce 模型通常要求每一轮的 Mapper/Reducer 操作后，数据必须落地存储到分布式文件系统中(改造后可存放到数据库中，如阿里云的 MaxComputer 表)，造成冗余的 I/O 操作。笔者提出一种面向树型结构的 Mapper/Chain Reducer 归并处理模型，一次 Map，基于树型结构的深度为控制次数的多次 Reduce，实现树型结构数据的分布式归并处理。每一次 reducer 的结果根据当前处理结点的最大深度进行相关处理后提交给下一次 reducer 操作，直到所有的输出结果记录的深度均到 1 为止，由此构成一个 reducer 链。其处理流程如图 4 所示。

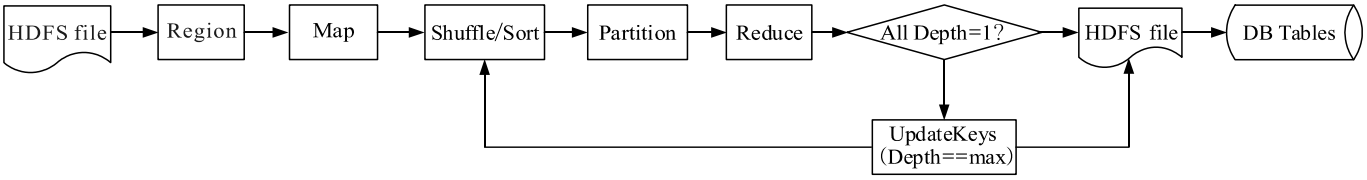

图 4　面向树型结构的 Mapper/Chain Reducer 归并处理模型

模型首先接收由数据库按照组合关键字-键值对特定要求(keys 中含有一个代表结点深度 Depth

的子 key)导出的数据文件，根据 MapReduce 模型的一般要求进行文件分片，执行 Map 操作，mapper 结果进行 shuffle 排序并分区，再执行 Reduce 操作，判断其输出结果数据集记录的所有 Depth 子键值是否均为 1，是则表明归并结束，输出结果至相应 HDFS 文件或数据库对应表中；如果不是，则执行 UpdateKeys 操作。

UpdateKeys 操作构成 Chain Reducer 的关键，其处理的对象是上一次 Reducer 的结果数据集。该数据集的数据将被 UpdateKeys()分为 2 部分：1) Depth 小于 Max(Depth)的所有记录，这部分数据不处理，原样保留；2) 子键 Depth 等于 Max(Depth)的记录，这些数据需要完成 2 个操作：一是输出到最终的结果集(文件或数据库)中，代表各层结点的归并结果；二是对其组合关键字中代表结点和深度的信息进行调整更新。更新操作具体为：将当前结点的信息替换为其父结点的信息，深度 Depth−1。更新后的数据集与 1)一起，送至 Shuffle/Sort 继续执行后续操作。

3.2 性能分析

实验采用完全分布式模式实现 Hadoop Map/Reduce 分布式处理架构，利用 1 台服务器中的 2 个虚拟机分别指定为 NameNode 和 JobTracker，构成 masters 主机集；利用 4 台高性能台式计算机作为 DataNode，每台物理机可创建不少于 4 个虚拟机作为 TaskTracker，构成 slaves 从机集。

根据文献[4]给出了评价 Map/Reduce 分布式计算模型算法性能的“加速比”的计算如式(3)所示，t_1、t_n分别表示在 1 个节点和 n 个节点上的执行时间。

$$a_n=t_1/t_2。 \tag{3}$$

在 4 台 DataNode 物理机的虚拟机 TaskTracker 数分别设置为 1, 2, 3, 4, 5 构成 4, 8, 12, 16 和 20 TaskTracker，取得的结果如图 5 所示。当单机运行 2 或 3 个虚拟机时，分布式归并模型加速比的增幅较为明显；当虚拟机数设置为 4 时仍有增长，但增幅下降；虚拟机个数为 5 个时增幅趋于平缓。主要原因是物理机上运行虚拟机数量越大，则系统负载越高导致性能瓶颈。如果虚拟机调整为单独的物理机运行 TaskTracker，则可获得更好的加速比。

文献[5]给出了评价算法性能“可扩展性”的计算公式如式(4)所示，w、p 表示问题规模和集群规模，w'、p' 表示扩展后的问题规模和集群规模。

$$\mathrm{Kcal}(w,w')=(w/p)/(w'/p')。 \tag{4}$$

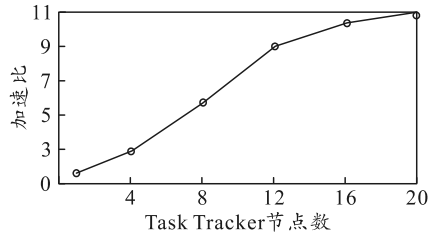

图 5 Map/Reduce 分布式归并模型的加速比

为了便于测试算法的可扩展性，与测试加速比方法相同，对其装备普查数据进行归并处理。4 台 DataNode 物理机的虚拟机 TaskTracker 数分别设置为 1, 2, 3, 4, 5 构成 4, 8, 12, 16 和 20 TaskTracker，实验结果如图 6 所示。在物理机增加虚拟机构成 4～12 个 TaskTracker 时，可扩展性能力增加明显；当增加到 16 时，可扩展性能力增幅明显减小；至 20 个 TaskTracker 时变化已不明显。实验结果总体表明，Map/Reduce 分布式归并模型能够取得较好的可扩展性。

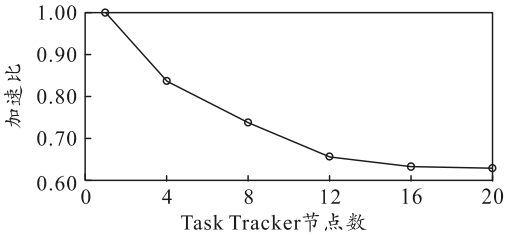

图 6 Map/Reduce 分布式归并模型的可扩展性

4 结束语

笔者针对装备维修数据的特点，提出基于单位层次树的数据归并模型与算法，对 MapReduce 并行计算模型进行改进，使其适用于关联性强的树型结构数据，通过实例进行实验，实现面向单位层次树的属性约简与数据归并，并分析 MapReduce 分布式模型的加速比和可扩展性。结果表明模型取得了较好的效果。

参考文献：

[1] 宋建社. 装备维修信息化工程[M]. 北京: 国防工业出版社, 2005: 15−30.

[2] 胡文瑜, 孙志挥, 吴英杰. 数据挖掘取样方法研究[J]. 计算机研究与发展, 2011, 9(1): 351−356.

[3] 李远方, 贾时银, 邓世昆, 等. 基于树结构的 MapReduce 模型[J]. 计算机技术与发展, 2011, 21(8): 149−152.

[4] ZHAO W, MA H, HE Q. Parallel K-Means Clustering Based on MapReduce[C]. IEEE International Conference on Cloud Computing, 2009: 674−679.

[5] 陈军, 莫则尧. 大规模并行应用程序的可扩展性研究[J]. 计算机研究与发展, 2000, 37(11): 1382−1388.