基于 MapReduce 的高效用序列模式挖掘算法①

程思远, 马 超, 李聪聪

(复旦大学 计算机科学技术学院, 上海 201203)
(上海市数据科学重点实验室(复旦大学), 上海 201203)

摘 要: 由于数据规模的快速增长, 高效用序列模式挖掘算法效率严重下降. 针对这种情况, 提出基于 MapReduce 的高效用序列模式挖掘算法 HusMaR. 算法基于 MapReduce 框架, 使用效用矩阵高效地生成候选项; 使用随机映射策略均衡计算资源; 使用基于领域的剪枝策略来防止组合爆炸. 实验结果表明, 在大规模数据集下, 算法取得了较高的并行效率.

关键词: 序列模式; MapReduce; 剪枝策略; 高效用序列模式挖掘; 随机策略

High Utility Sequential Pattern Mining Algorithm Based on MapReduce

CHENG Si-Yuan, MA Chao, LI Cong-Cong

(School of Computer Science, Fudan University, Shanghai 201203, China)
(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract: Because of the rapid growth of data, the high utility sequential pattern mining algorithms' efficiency decreases seriously. In view of this, we propose a high utility sequential pattern mining algorithm based on MapReduce, namely HusMaR. This algorithm is based on MapReduce, which using the utility matrix to generate candidate efficiently, random mapping strategy to balance of computing resources and field-based pruning strategy to prevent an explosion. Experimental results show that in the large scale of data, the algorithm achieves a high parallel efficiency.

Key words: sequential pattern; MapReduce; pruning strategy; high utility sequential pattern mining; random strategy

在传统的序列模式挖掘应用中, 是将支持度(频次)作为选择序列模式的基准[1-3]. 但是在实际应用中, 这种基于支持度框架的序列模式挖掘算法得出的结果往往由于结果数量多且相似度大而导致信息量缺失, 难以给出相关领域解释[4]. 以零售业为例, 卖出汽车的利润远大于卖出牛奶的, 但是由于牛奶的销量远大于汽车的, 若商家希望得到高利润的序列模式用于商业决策, 那么基于支持度框架的序列模式挖掘算法将无法处理这类问题[4]. 为了解决如何挖掘用户感兴趣的序列模式的问题, 效用值(利润)被引入作为选择序列模式的基准[4].

现今, 随着数据规模的快速增长, 无论是基于支持度框架的序列模式挖掘算法(如 SPADE 算法[1], PrefixSpan 算法[2]和 SPAM 算法[3])或是基于效用值框架的高效用序列模式挖掘算法(如 USpan 算法[4], UMSP 算法[5], UI 算法[6]和 US 算法[6]), 当面对大规模数据集的输入时, 算法的运行效率会严重下降, 并且在大数据环境下, 单机将无法完成序列模式挖掘算法的计算任务. 面对大数据环境和单机性能瓶颈的问题, 目前已有一些使用 MapReduce 框架[7]实现的基于支持度框架的序列模式挖掘算法的研究成果[8,9]. 但是使用 MapReduce 框架实现基于效用值框架的高效用序列模式挖掘算法却没有人进行研究. 主要原因是基于效用值框架的高效用序列模式挖掘算法与基于支持度框架的序列模式挖掘算法不同, 它不再保持 downward closure property[10]属性, 序列化序列元素会导致算法面临组合爆炸和计算复杂度的问题[4]. 因此在大数据环境下, 上述问题将转为存储和计算资源的问题:

(1)存储问题: 文献[4,5]使用树结构存储产生的所有序列候选项, 但是当面对巨量的序列数据时, 我们无

① 收稿时间:2015-04-07;收到修改稿时间:2015-05-12

法将完整的树结构存储在内存中.

(2)计算资源: 为了防止组合爆炸, 在产生新的序列候选项时, 会采用剪枝策略[4,6], 这种产生新的序列候选项和剪枝策略的过程需要扫描全部数据, 这部分扫描的计算消耗巨大.

本文主要的贡献是解决了如何在大数据环境下使用 MapReduce 框架挖掘高效用序列模式的问题. 本文采用效用矩阵、随机映射策略和剪枝策略实现了一种基于 MapReduce 的高效用序列模式挖掘算法 HusMaR(High Utility Sequential Pattern Mining Algorithm Based on MapReduce), 通过在仿真数据集上进行效率实验, 结果表明算法具有较高的并行性.

1 问题定义

在高效用序列模式挖掘中, 为了描述问题, 将序列模式中的项、项集、序列、序列数据库扩展为 q-项、q-项集、q-序列和 q-序列数据库. 例如表 2 为 q-序列数据库的样例, 正整数表示对应项的数量, 称(e,5)为 q-项, 记为(i,q), 称[(c,2)(f,1)]为 q-项集, 记为 l, 称<(e,5)[(c,2)(f,1)](b,2)>为 q-序列, 记为 s.

基于表 1 的项的利润值映射表, 单个项的利润值等于 $p(i)$. 假设 $f_{u_i}, f_{u_l}, f_{u_s}$ 分别为 q-项的效用值, q-项集的效用值和 q-序列的效用值的计算公式, 记为 $u(i,q), u(l), u(s)$, 计算公式一般由领域专家给出[4].

表 1 利润映射表[4]

item	a	b	c	d	e	f
quality	2	5	4	3	1	1

定义 1.[4] 令 $u(i,q)=p(i)\cdot q$, $u(l)$ 为 l 中所有 $u(i,q)$ 的和, $u(s)$ 为 s 中所有 $u(l)$ 的和. 对于序列 t, 它的序列效用值可形式化地表示为 $u_{\max}(t)=\sum \max\{u(s')|s':t\wedge s'\subseteq \wedge s\in S\}$.

定义 2.[4] 若给定效用值的阈值 ξ, 当且仅当 $u_{\max}(t)\geq\xi$ 时, 称序列 t 为高效用序列模式.

表 2 q-序列数据库[4]

sid	q-sequence
1	<(e, 5)[(c, 2)(f, 1)](b, 2)>
2	<[(a, 2)(e, 6)][(a, 1)(b, 1)(c, 2)][(a, 2)(d, 3)(e, 3)]>
3	<(c, 1)[(a, 6)(d, 3)(e, 2)]>
4	<[(b, 2)(e, 2)][(a, 7)(d, 3)][(a, 4)(b, 1)(e, 2)]>
5	<[(b, 2)(e, 3)][(a, 6)(e, 3)][(a, 2)(b, 1)]>

序列<ea>为例, 它在表 2 的 q-序列数据库中匹配的效用值分别为{{ }, {8,10}, { }, {16,10}, {15, 7,7}}. { }表示序列<ea>与 s_1 未匹配成功, {8, 10}表示序列<ea>与 s_2 匹配成功 2 次, s_2 中第一个项集中的 e 分别和第二、第三个项集中的 a 组合而成. 依此类推, 根据公式可得 $u_{\max}(<ea>)=10+16+15=41$.

2 HusMaR算法

效用矩阵用于过滤无用的单项序列、产生序列候选项; 随机映射策略用于均衡计算量; 基于领域的剪枝策略用于过滤序列候选项.

本文使用上述三种思想克服大数据环境下存储和计算资源的问题. 下文详细描述效用矩阵、随机映射策略和基于领域的剪枝策略.

2.1 效用矩阵

表 3 为 q-序列为<[(b, 2)(e, 2)][(a, 7)(d, 3)][(a, 4)(b, 1)(e, 2)]>的效用矩阵.

表 3 q-序列效用矩阵[4]

items	q-itemset 1	q-itemset 2	q-itemset 3
a	(0, 50)	(14, 24)	(8, 7)
b	(10, 40)	(0, 24)	(5, 2)
d	(0, 40)	(9, 15)	(0, 2)
e	(2, 38)	(0, 15)	(2, 0)

表 3 中第 1 行第 1 列(0,50)这一元组, 0 表示 a 在 q-itemset 1 中的效用值, 因为 a 没有出现, 所以为 0, 50 代表去除 a 和 a 之前的所有项后, q-序列剩余项的效用值之和.

定义 3. 给定序列 t 只含一个项 i, 那么序列 t 的可用效用值可表示为 $u_{remain}(t)=\sum_{s\in S}(u_{rest}(i,s)+u(i,q))$, $u_{rest}(i,s)$ 为项 i 对 q-序列 s 的最大剩余效用值, $u(i,q)$ 为取得最大剩余效用值时, 项 i 的效用值. $u_{remain}(t)$ 为单项序列 t 效用值的上界.

例如单项序列<b>, 它在表 2 中所对应的 $u_{rest}(i,s)+u(i,q)$ 为 {10,29,0,50,37}, $u_{remain}(t)$ 为 126.

定义 4. 给定效用值的阈值 ξ, 当且仅当 $u_{remain}(t)\geq\mu\cdot\xi$, $\mu\geq1$ 时, 称单项序列 t 为可用的单项序列, μ 称为松弛率.

本文设置 $\mu=1$, 所以若不是以 t 为起始的序列模式, 那么这些序列模式结果不可能是高效用序列模式.

通过 MapReduce 过程, 采用效用矩阵可快速提取可用的单项序列. 利用可用的单项序列集合, 过滤 q-

序列数据库中的单项序列, 避免无用的单项序列产生候选项时带来系统资源的消耗和算法效率的降低. 候选项的产生可以通过效用矩阵、项集内拼接和序列间拼接完成[4].

2.2 随机映射策略

为了减小单个节点的资源消耗, 充分利用集群的计算能力, 本文基于 Random(*K*)的随机算法为每一个 q-序列分配键值, 均衡地将 q-序列数据库中所有的 q-序列分组, 以降低存储和计算资源的消耗, 均衡地为集群中每个节点分配任务数量. 合理地分配分组中 q-序列的数量, 不仅可以加快算法执行效率, 同时还可以防止单个计算节点出现内存溢出的现象.

2.3 剪枝策略

(1)基于序列结构复杂度的剪枝, 拼接的过程中, 拼接效用值高且剩余效用值高的 M 个项.

序列候选项的结构复杂度和尺度与 q-序列相关. 复杂的 q-序列会产生很多的序列候选项, 降低查询的效率, 同时结构复杂的序列候选项得出的序列模式不易于解释. 本文基于实际应用中结构简单的模式易于解释的领域经验, 通过控制 M 的大小来控制序列模式结构的复杂度.

(2)基于 q-序列的尺度的剪枝, 若候选项的尺度达到 N, 则停止拼接新项.

当给定一个 q-序列数据库, 最大尺度为 L. 挖掘所有尺度小于等于 L 的候选项集合为C_L, 则C_L的容量为$|C_L|$, 所需时间为T_L; 挖掘所有尺度小于等于 N 的候选项集合为C_N, 则C_N的容量为$|C_N|$, 所需时间为T_N. 因为$T_N < T_L$, 令$\lambda = |C_L| - |C_N| / |C_L|$, 若$\lambda < \beta, \beta \in (0,1)$成立, 则称 N 为算法可接受的序列模式长度, 其中λ为算法模式丢失率, β为算法容忍率. 该策略以准确性换时间的折中方法来提高算法执行效率.

这两种剪枝策略组合起看, 通过限制条件挖掘出特定的序列模式, 并且序列模式结果集是可接受的.

2.3 MapReduce 过程

基于序列效用值的定义可知, 序列 t 的效用值与 q-序列数据库 S 里的每个 q-序列 s 有关, 若序列 t 匹配到 q-子序列s', 取$u_{\max}(s')$, 最后累加得到序列 t 的$u_{\max}(t)$. 这种批处理式的计算模式非常适合使用 MapReduce 框架. 整个过程采用逆向实现, q-序列 s 使用效用矩阵产生序列候选项, 得到所有的 q-子序列s'和$u_{\max}(s')$, q-子序列s'反向替代序列 t, 累加所有的$u_{\max}(s')$, 得到$u_{\max}(t)$.

图 1 中给定输入 q-序列数据库 S, 经过三次 MapReduce 过程并行计算高效用序列模式, 最后将结果输出到文件系统.

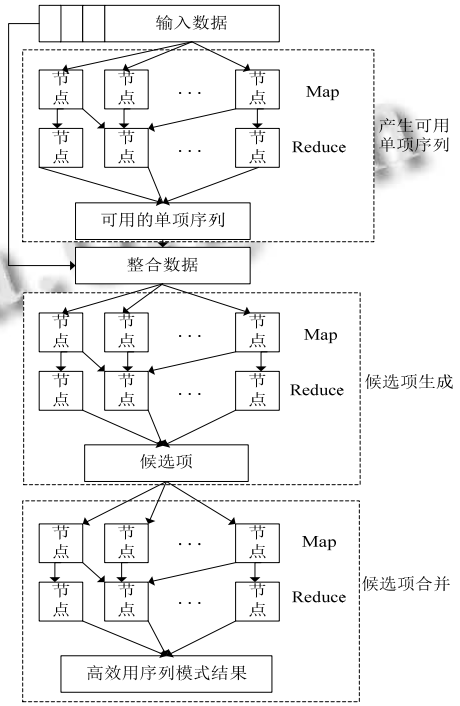

图 1　算法 MapReduce 框架图

步骤 1: 可用的单项序列生成: 将 S 分片, 每个 Map 将输入的 q-序列 s 作为 value, 构建 value 的效用矩阵, 得出每个单项序列 t 的 $u_{rest}(i,s)+u(i,q)$, 输出为 $<t,u_{rest}(i,s)+u(i,q)>$. 在 Reduce 端, 输入为 <t,List< $u_{rest}(i,s)+u(i,q)$>>, 合并 List 中的值, 得到 t 的可用效用值, 若大于阈值ξ, 则输出该单项序列 t.

```
Mapper<key, value=s>
foreach t in UtilityMatrix(s) do
        Call Output(t, u_rest(i,s)+u(i,q))
end
Reducer<key=t, value=List< u_rest(i,s)+u(i,q)>>
uRemaining=0
foreach u in value do
        uRemaining=u+ uRemaining
end
if uRemaining > ξ then
        Call Output(null, t)
end
```

图 2　步骤 1 可用单项序列生成算法

步骤 2: 候选项生成: 将 S 分片, 每个 Map 将输入的 q-序列 s 作为 value, 用随机映射策略确定 s 的键值作为 Map 输出的键, s 作为输出的值; 这样 Reduce 得到的输入<outkey,List<s>>. Reduce 中利用可用单项序列信息, 结合 UtilityMatrix(s)的方法可生成相应 s 经过剪枝后的候选项<s',$u_{\max}(s')$>的集合, 最后合并输出到文件系统.

```
Mapper<key, value=s>
outkey=Random(K)
Call Output(outkey,value)
Reducer<key=outkey, value=List<s>>
Result=Set()
foreach s in value do
    Candidates get from UtilityMatrix(s)
    Merge Candidates into Result
end
Call Output(null, Result)
```

图 3 步骤 2 候选项生成算法

步骤 3: 合并候选项: 步骤 2 中S生成了很多候选项, 候选项可表示为<t,$utility$>, 每个 Map 以此为输入, 经过 Map 后形成输出<t,List<$utility$>>作为 Reduce 过程的输入. 根据 $u_{max}(t)$式子, 输出结果为<t,$u_{max}(t)$>的形式到文件系统中.

```
Mapper<key, value=Result>
foreach <t, utility> in value do
    Call Output(t, utility)
end
Reducer<key=t, value=List<utility>>
U=0
foreach u in value do
    U=u+ U
end
if U > ξ then
    Call Output(t, U)
end
```

图 4 步骤 3 合并候选项算法

3 实验

实验使用 3 个仿真数据集 DS1(序列数 10k), DS2(序列数 100k)和 DS3(序列数 1000k). 集群配置: 5 台虚拟机. 虚拟机配置: 内存 16GB, 硬盘 300GB, CPU 为 Intel E5649. 算法均为 Java 实现.

算法参数有 K, ξ, M 和 N. ξ为效用值的阈值, 我们设置$\xi = 10$; M 为经验取值, 在实际应用中, 我们发现通常项集一般都在 5 以下, 即 M=5 时, 序列模式的覆盖率已足够高, M > 5 时, 虽然会得到项集为 5 的模式, 但是会大幅降低查询效率, 而且这样的模式又是在实际应用中非常少的. 因此, 我们建议设置 M <= 5.

有关 K 取值问题, 与集群的计算节点数有关, 节点数一定的情况下, K 越大, 平均的节点计算效率越高. 如图 5, 控制 M=2, N=2 时, 我们可以发现算法时间确实是随着 K 增大而降低, 但是当 K 趋近 10000 时, 时间下降的趋势变得缓和, 因此考虑实验环境, 我们最终设置 K=10000.

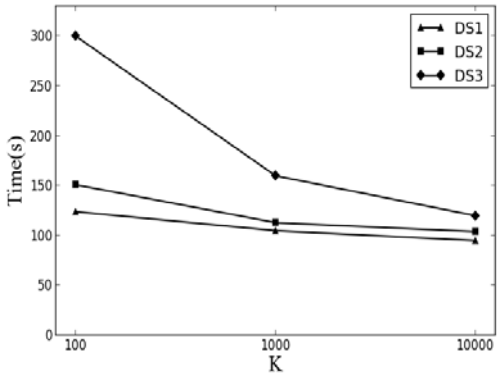

图 5 K 与 HusMaR 效率关系

在三个仿真数据集中, 序列的尺度最大为 15, 平均每个序列元组有 3 个项, 平均序列尺度为 10. 当 M=2 时, 从图 6 中可以看出, 当 N 取值为平均序列尺度时, 丢失率是可以接受的.

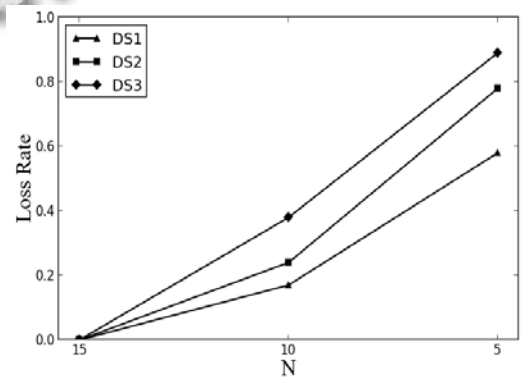

图 6 N 与丢失率关系图

图 7 对比了本文算法的单机版本和 MapReduce 版本在仿真数据集下效率的表现. 从实验结果可以看出,

在数据量等于 10k 时, 单机算法执行时间略优于 MapReduce 算法执行时间, 而当数据量等于 100k 时, 单机算法执行时间已远远大于 MapReduce 算法的执行时间, 当数据量等于 1000k 时, 单机算法已经无法运行, 而 MapReduce 算法仍可以正常运行, 且具有较高的效率.

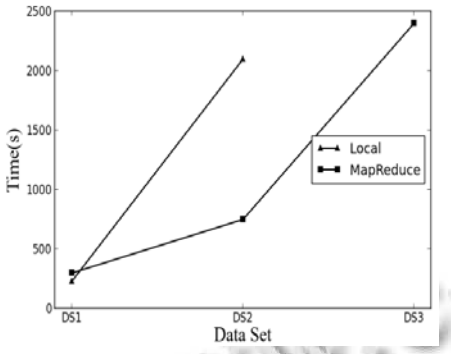

图 7 单机算法与并行 HusMaR 效率对比

4 结语

本文使用 MapReduce 框架,采用效用矩阵、随机映射策略和基于领域的剪枝策略, 实现了一种基于效用值的高效用序列模式挖掘算法 HusMaR. 通过在仿真数据集上的实验, 表明算法在大数据环境下并行的高效性. 我们后续的工作将是优化剪枝策略, 更好地解决算法效率、序列模式结构复杂性和序列尺度之间平衡的问题.

参考文献

1 Zaki MJ. SPADE: An efficient algorithm for mining frequent sequences. Machine Learning, 2001, 42(1-2): 31–60.

2 Pei J, Pinto H, Chen Q, et al. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. IEEE 29th International Conference on Data Engineering (ICDE). IEEE Computer Society, 2001.

3 Ayres J, Flannick J, Gehrke J, et al. Sequential pattern mining using a bitmap representation. Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2002: 429–435.

4 Yin J, Zheng Z, Cao L. Uspan: an efficient algorithm for mining high utility sequential patterns. Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2012: 660–668.

5 Shie BE, Hsiao HF, Tseng VS, et al. Mining high utility mobile sequential patterns in mobile commerce environments. Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2011: 224–238.

6 Ahmed CF, Tanbeer SK, Jeong BS. A novel approach for mining high-utility sequential patterns in sequence databases. ETRI Journal, 2010, 32(5):676–686.

7 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107–113.

8 Wei Y, Liu D, Duan L. Distributed PrefixSpan algorithm based on MapReduce. 2012 International Symposium on Information Technology in Medicine and Education (ITME). IEEE, 2012, 2: 901–904.

9 Chen CC, Tseng CY, Chen MS. Highly scalable sequential pattern mining based on MapReduce model on the cloud. 2013 IEEE International Congress on Big Data. IEEE, 2013: 310–317.

10 Agrawal R, Srikant R. Mining sequential patterns. Proc. of the 11th Int. Conf. on Data Engineering, 1995. IEEE, 1995: 3–14.

11 Fournier-Viger P, Wu CW, Gomariz A, et al. VMSP: Efficient Vertical Mining of Maximal Sequential Patterns. Advances in Artificial Intelligence. Springer International Publishing, 2014: 83–94.

12 Yin J, Zheng Z, Cao L, et al. Efficiently mining top-K high utility sequential patterns. 2013 IEEE 13th International Conference on Data Mining,. IEEE, 2013: 1259–1264.