深入理解大数据-大数据处理与编程实践

Ch 9. 数据挖掘基础算法

南京大学计算机科学与技术系

主讲人：黄宜华，顾荣

鸣谢：本课程得到Google（北京）与Intel公司
中国大学合作部精品课程计划资助

Ch 9. 数据挖掘基础算法

9.1 数据挖掘并行算法研究的重要性

9.2 基于MapReduce的K-Means聚类算法

9.3 基于MapReduce的分类算法

9.4 基于MapReduce的频繁项集挖掘算法

9.1 数据挖掘并行算法研究的重要性

- 数据挖掘是通过对**大规模观测数据集**的分析，寻找隐藏在这些数据集中的有用信息和事实的过程。

- 数据挖掘的特征之一：海量数据

  Small data does not require data mining, large data causes problems —— 摘自黎铭的《数据挖掘》课件

- 研究发现大数据隐含着更为准确的事实

- 因此海量数据挖掘是并行计算中值得研究的一个领域

研究发现大数据隐含着更为准确的事实

2001年微软研究院的Banko and Brill*等研究发现数据越大，机器学习的精度越高；当数据不断增长时，不同算法的分类精度趋向于相同！

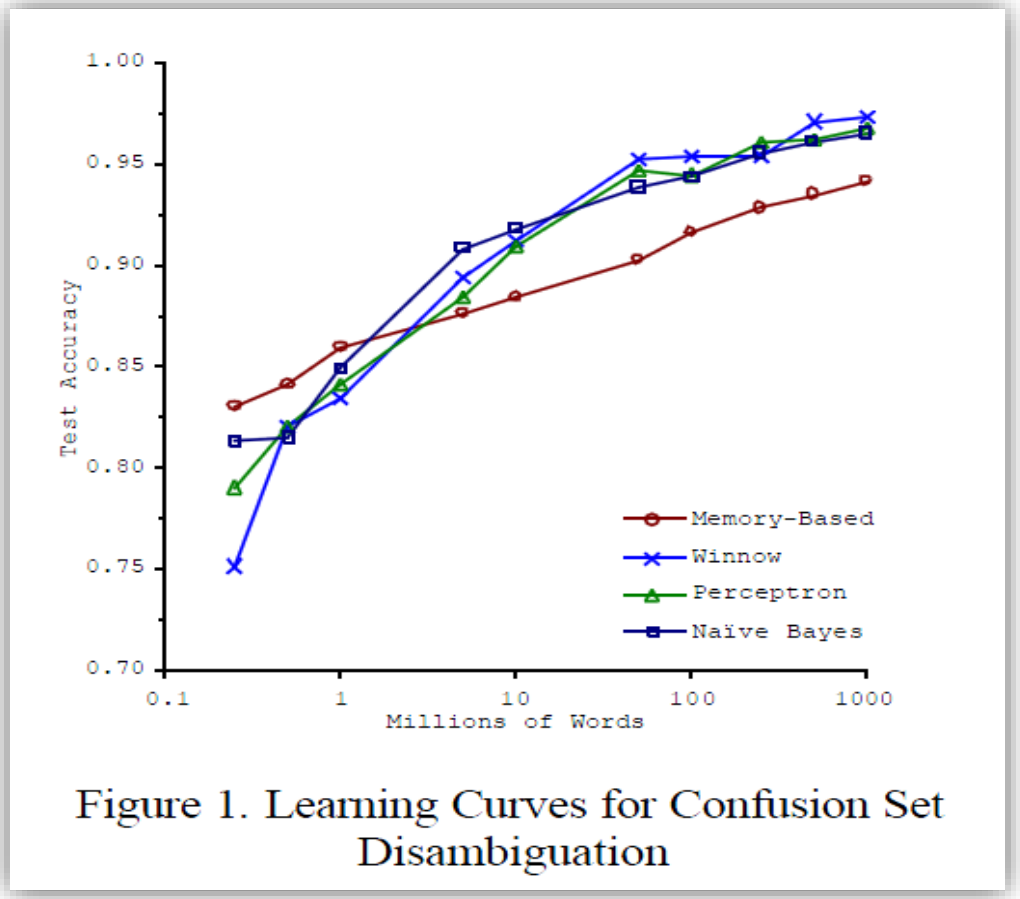


Figure 1. Learning Curves for Confusion Set Disambiguation

* M. Banko and E. Brili (2001). **Scaling to very very large corpora for natural language disambiguation**. *ACL 2001*

研究发现大数据隐含着更为准确的事实

2007年Google公司Brants等基于MapReduce研究了一个2万亿单词训练数据集的语言模型，发现大数据集上的简单算法能比小数据集上的复杂算法产生更好的结果

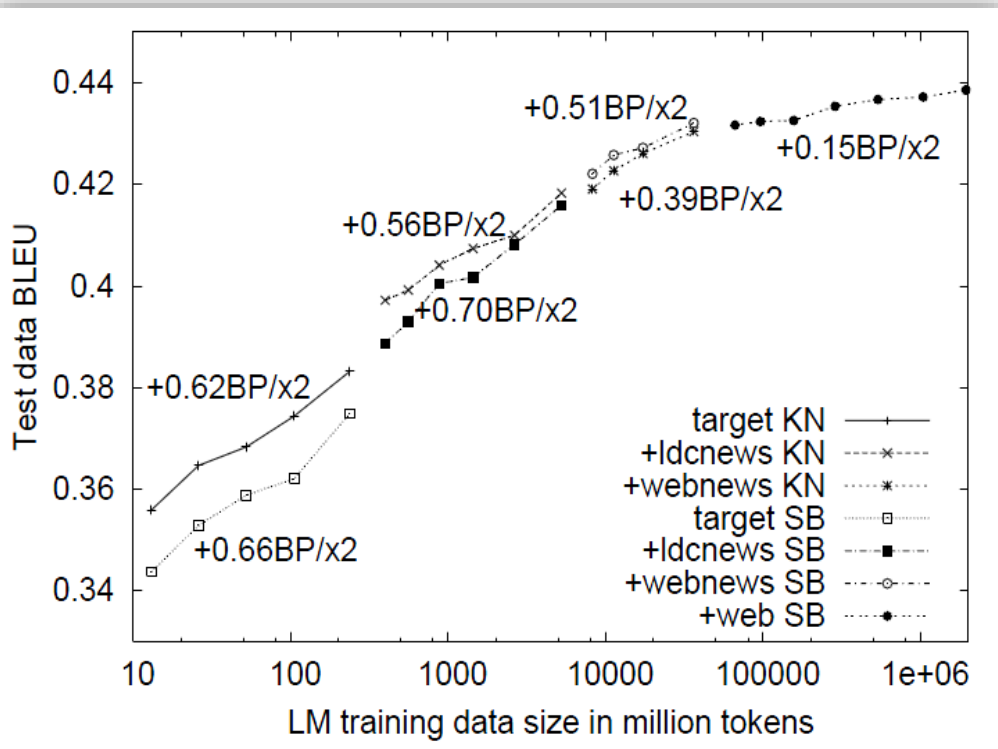


Figure 5: BLEU scores for varying amounts of data using Kneser-Ney (KN) and Stupid Backoff (SB).

* T. Brants, A. C. Popat, et al. **Large Language Models in Machine Translation**. In EMNLP-CoNLL 2007 - Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning

Ch 9. 数据挖掘基础算法

9.1 数据挖掘并行算法研究的重要性

9.2 基于MapReduce的K-Means聚类算法

9.3 基于MapReduce的分类算法

9.4 基于MapReduce的频繁项集挖掘算法

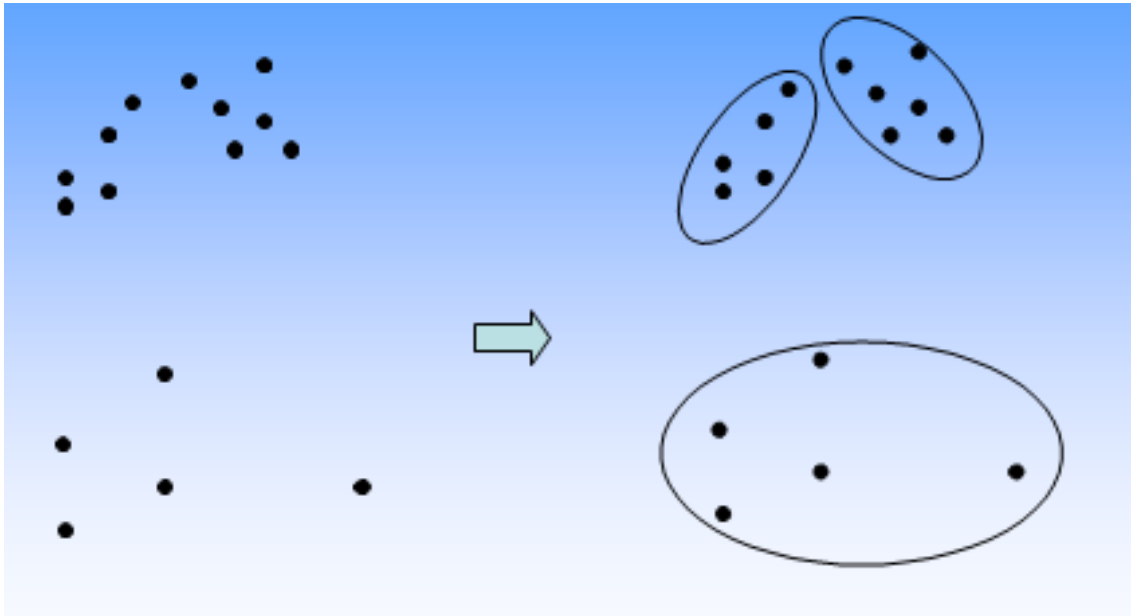
9.2 基于MapReduce的K-Means聚类算法

1. K-Means聚类算法介绍
2. 基于MapReduce的K-Means并行算法设计
3. 实验结果与小结
4. 聚类算法应用实例

1. K-Means聚类算法介绍

- **定义**：将给定的多个对象分成若干组，组内的各个对象是相似的，组间的对象是不相似的。进行划分的过程就是聚类过程，划分后的组称为簇(cluster)。

- 几种聚类方法：
  - 基于划分的方法；
  - 基于层次的方法；
  - 基于密度的方法；
  - … …



数据点的数值类型

- 数据点的类型可分为：
  - 欧氏（Euclidean）
  - 非欧

- 这二者在数据的表示以及处理上有较大的不同：
  - 怎样来表示cluster ？
  - 怎样来计算相似度

Cluster的表示

- 欧氏空间：
  - 取各个数据点的平均值（centroid）
- 非欧空间
  - 取某个处于最中间的点
  - 取若干个最具代表性的点（clustroid）
  - ... ...

相似度(距离)的计算

- 欧氏空间：可以有较为简单的方法

$$d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- 非欧氏空间：通常不能直接进行简单的数字计算
  - Jaccard 距离: 1 - |S ∩ T| / | S ∪ T|
  - Cosine距离：两个向量的夹角大小
  - Edit 距离：适合于string类型的数据

基于划分(Partitioning)的聚类方法

- 给定N个对象，构造K个分组，每个分组就代表一个聚类。

- K个分组满足以下条件：
  - 每个分组至少包含一个对象；
  - 每个对象属于且仅属于一个分组；

- K-Means算法是最常见和典型的基于划分的聚类方法

K-Means算法

输入：待聚类的N个数据点，期望生成的聚类的个数K
输出：K个聚类
算法描述:

选出K个点作为初始的cluster center
Loop:
对输入中的每一个点p：
{
计算p到各个cluster的距离；
将p归入最近的cluster;
}
重新计算各个cluster的中心

如果不满足停止条件，goto Loop; 否则，停止

过程示例

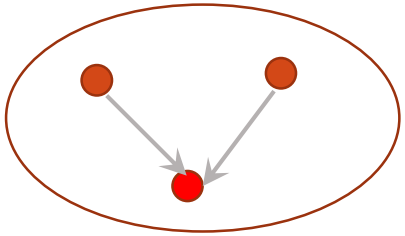
初始数据
K = 2

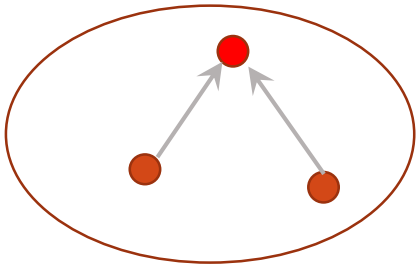
选择初始中心

第1次聚类：计算距离

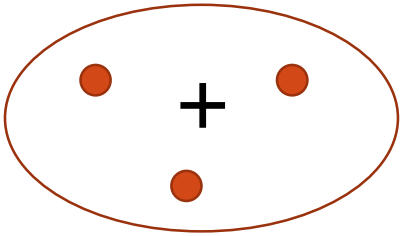
过程示例(cont.)

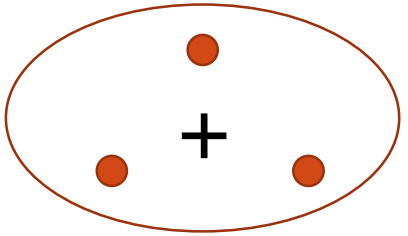
第1次聚类：
归类各点





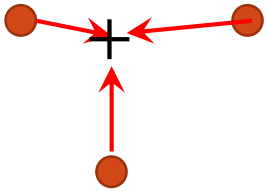
重新计算聚类中心

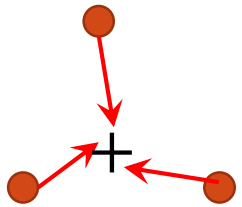




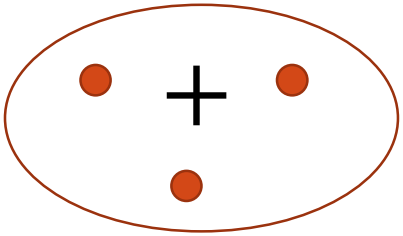
过程示例(Cont.)

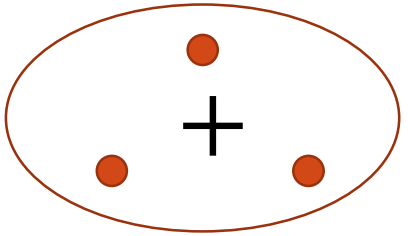
第2次聚类：
计算距离





第2次聚类：
归类各点





聚类无变化，迭代终止

K-Means是个不断迭代的过程

- 第 i轮迭代：
  - 生成新的clusters，并计算cluster centers
- 第 i+1轮迭代：
  - 根据第i轮迭代中生成的clusters和计算出的cluster centers，进行新一轮的聚类
- 如此不断迭代直到满足终止条件

K-Means算法的局限性

- 对初始cluster centers的选取会影响到最终的聚类结果
- 由此带来的结果是：能得到局部最优解，不保证得到全局最优解
- 相似度计算和比较时的计算量较大

K-Means计算性能的瓶颈

- 如果样本数据有n个，预期生成k个cluster，则K-Means算法t次迭代过程的时间复杂度为O(nkt)，需要计算ntk次相似度
- 如果能够将各个点到cluster center相似度的计算工作分摊到不同的机器上并行地计算，则能够大幅提高计算速度
- 本课将介绍利用MapReduce来将K-Means聚类过程并行化

2. 基于MapReduce的K-Means并行算法设计

考虑数据相关度

- 在进行K-Means聚类中，在处理每一个数据点时
- 只需要知道各个cluster 的中心信息
- 不需要知道关于其他数据点的任何信息

◈ 所以，如果涉及到全局信息，只需要知道关于各个cluster center的信息即可

MapReduce并行化聚类算法设计思路

- 将所有的数据分布到不同的MapReduce节点上，每个节点只对自己的数据进行计算
- 每个Map节点能够读取上一次迭代生成的cluster centers，并判断自己的各个数据点应该属于哪一个cluster
- Reduce节点综合每个属于每个cluster的数据点，计算出新的cluster centers

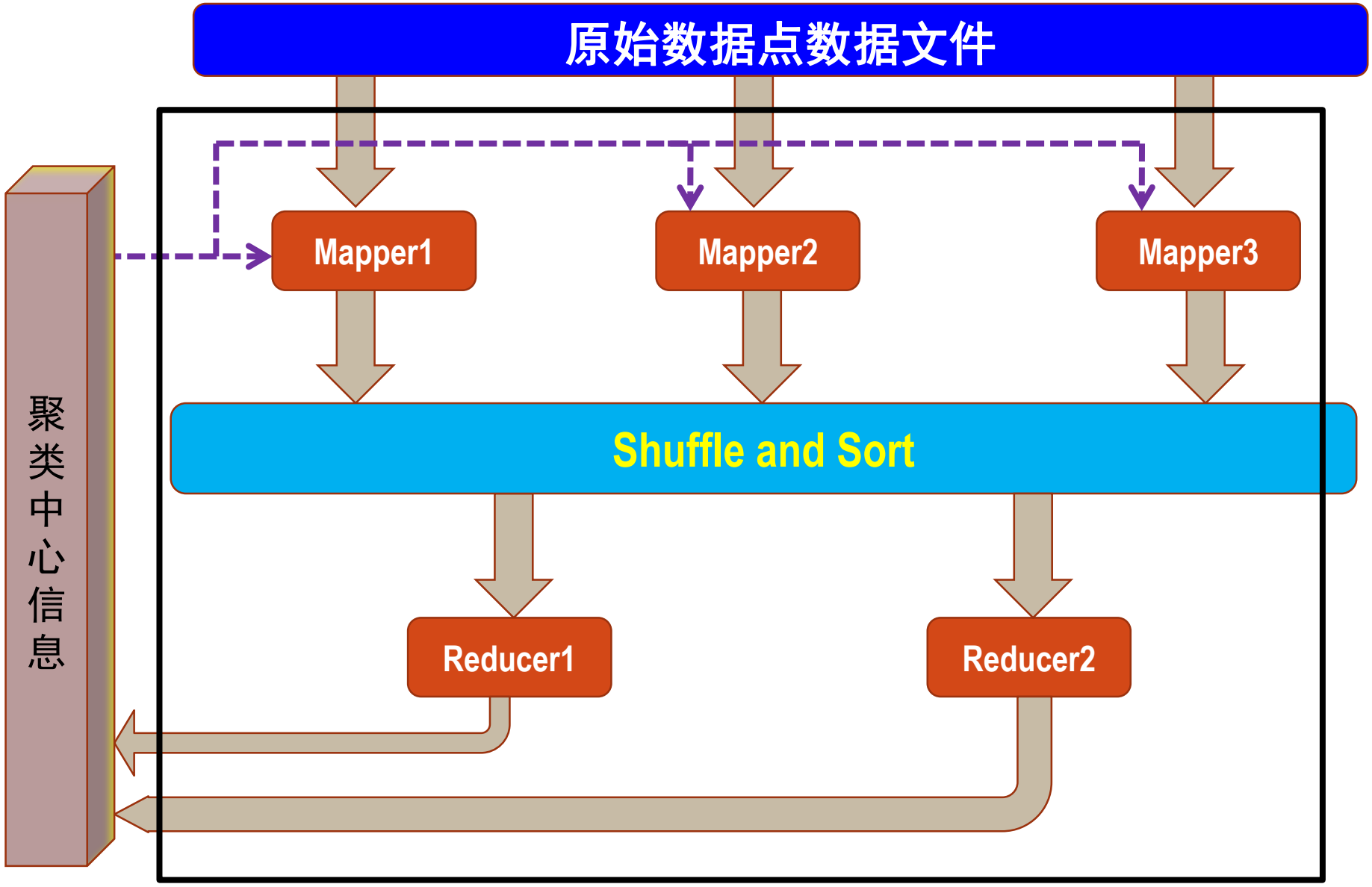
需要全局共享的数据

- 每一个节点需要访问如下的全局文件
  - 当前的迭代计数
  - K个表示不同聚类中心的如下的数据结构

cluster id
cluster center
属于该cluster center的数据点的个数

- 这是唯一的全局文件

基于MapReduce的K-Means并行算法设计



Map阶段的处理

- 在Map类的初始化方法setup中读取全局的聚类中心信息
- 对Map方法收到的每一个数据点p，计算p与所有聚类中心间的距离，并选择一个距离最小的中心作为p所属的聚类，输出<ClusterID,(p,1)>键值对

- 对每个Map节点上即将传递到Reduce节点的每一个<ClusterID,(p,1)>键值对，用Combiner进行数据优化，合并相同ClusterID下的所有数据点并求取这些点的均值pm以及数据点个数n

Map阶段的处理

Mapper伪代码

```
class Mapper
  setup(…)
  {
      读出全局的聚类中心数据 → Centers
  }

  map(key, p)  // p为一个数据点
   {
     minDis = Double.MAX VALUE;
     index = -1;
     for i=0 to Centers.length
     {
        dis= ComputeDist(p, Centers[i]);
        if dis < minDis
        {   minDis = dis;
            index = i;
        }
     }
     emit(Centers[i].ClusterID, (p,1));
  }
```

Map阶段的处理

Combiner伪代码

```
class Combiner
  reduce(ClusterID, [(p1,1), (p2,1), ...])
  {
     pm = 0.0;
     n = 数据点列表[(p1,1), (p2,1), ...]中数据点的总个数;
     for i=0 to n
         pm += p[i];
     pm = pm / n;  // 求得这些数据点的平均值
     emit(ClusterID, (pm, n));
  }
```

思考题：
在map中，最后一行emit(Centers[i].ClusterID, (p,1))语句中，为什么输出值必须是（p,1)而不能是p？如果写成p会带来什么问题？

Reduce阶段的处理

- 经过Map和Combine后从Map节点输出的所有ClusterID相同的中间结果<ClusterID, [(pm1, n1), (pm2, n3)…]>,计算新的均值pm，输出<ClusterID, pm>
- 所有输出的<ClusterID, (pm,n)>形成新的聚类中心，供下一次迭代计算

思考题答案：
用不用Combiner仅仅会影响性能，不能改变计算结果。因此，Combiner输出时不允许改变Map输出键值对中Value的格式和类型，否则会出错

Reduce阶段的处理

Reducer伪代码

```
class Reducer
  reduce(ClusterID, value = [(pm1,n1),(pm2,n2) ...])
  {
     pm = 0.0；   n=0;
     k = 数据点列表中数据项的总个数;
     for i=0 to k
     {    pm += pm[i]*n[i];  n+= n[i]; }
     pm = pm / n;  // 求得所有属于ClusterID的数据点的均值
     emit(ClusterID, (pm,n)); // 输出新的聚类中心的数据值
  }
```

MapReduce K-Means聚类处理示例

Object	Attribute 1 (x)	Attribute 2(y)
A	1	1
B	2	1
C	4	3
D	5	4

- 令 k = 2，欲生成 cluster-0 和 cluster-1
- 随机选取A(1,1)作为cluster-0的中心，C(4,3)作为cluster-1的中心
- 假定将所有数据分布到2个节点 node-0 和 node-1 上,即

  node-0 ： A(1,1)和C(4,3)

  node-1 ： B(2,1)和D(5,4)

MapReduce K-Means聚类处理示例

- 在开始之前，首先创建一个如前所述的全局文件

Iteration No	0	
Cluster id	cluster 中心	# of data points assigned
cluster -0	A(1, 1)	0
cluster-1	C(4, 3)	0

MapReduce K-Means聚类处理示例

Map阶段

- 每个节点读取全局文件，以获得上一轮迭代生成的cluster centers等信息
- 计算本节点上的每一个点到各个cluster center的距离，选择距离最近的cluster
- 为每一个数据点，emit
  <cluster assigned to , 数据点自身>

MapReduce K-Means聚类处理示例

Map阶段

- 计算各个数据点到各个cluster的距离，假定是如下的结果

	数据点	到各个 cluster 的距离比较		Assigned to
		cluster-0	cluster-1	
Node-0	A(1, 1)	近		cluster-0
Node-0	C(4,3)		近	cluster-1
Node-1	B(2,1)		近	cluster-1
Node-1	D(5,4)		近	cluster-1

node-0输出：
<cluster-0, [A(1,1),1]>
<cluster-1, [C(4,3),1]>

node-1输出：
<cluster-1, [B(2,1),1]>
<cluster-1, [D(5,4),1]>

MapReduce K-Means聚类处理示例

Map阶段的Combine处理

- 利用combiner合并map阶段产生的大量的ClusterID相同的键值对<ClusterID, [p,k)]>
- Combiner计算要emit的所有数据点的均值，以及这些数据点的个数
- 然后，为每一个cluster发射新的键值对<ClusterID, [pm, n]>

MapReduce K-Means聚类处理示例

Map阶段的Combine处理

- Map的输出（即Combiner的输入）：

node-0输出：
<cluster-0, [A(1,1),1] >
<cluster-1, [C(4,3) ,1]>

node-1输出：
<cluster-1, [B(2,1) ,1]>
<cluster-1, [D(5,4) ,1]>

- Combiner的输出
- key - ClusterID
- value - <ClusterID, (pm, n)>

node-0发射：
<cluster-0, [(1,1), 1]>
<cluster-1, [(4,3),1]>

node-1发射：
<cluster-1, [(3.5,2.5),2]>

MapReduce K-Means聚类处理示例

Reduce阶段

- 由于map阶段emit的key是clusterID，所以每个 cluster的全部数据将被发送同一个reducer，包括：
  - 该cluster 的ID
  - 该cluster的数据点的均值，及对应于该均值的数据点的个数

- 然后经计算后输出
  - ☞当前的迭代计数
  - ☞ClusterID
  - ☞新的Cluster Center
  - ☞属于该Cluster Center的数据点的个数

MapReduce K-Means聚类处理示例

Reduce阶段

- 在上一阶段，Combiner的输出

node-0发射：
<cluster-0，[(1,1), 1]>
<cluster-1，[(4,3),1]>

node-1发射：
<cluster-1，[(3.5,2.5),2]>

- 两个reducer分别收到

Reducer-0：
<cluster-0，[(1,1),1]>

Reducer-1：
<cluster-1,[1,(4,3),1]>
<cluster-1,[(3.5,2.5),2]>

MapReduce K-Means聚类处理示例

Reduce阶段

Reducer-0：
<cluster-0, [(1,1),1]>

Reducer-1：
<cluster-1,[1,(4,3),1]>
<cluster-1,[(3.5,2.5),2]>

- 计算可得
- cluster-0的中心 = (1, 1)
- cluster-1的中心 = $\left(\frac{4+2\times3.5}{1+2}, \frac{3+2\times2.5}{1+2}\right)$

= (3.67, 2.67)

MapReduce K-Means聚类处理示例

- Reducer输出

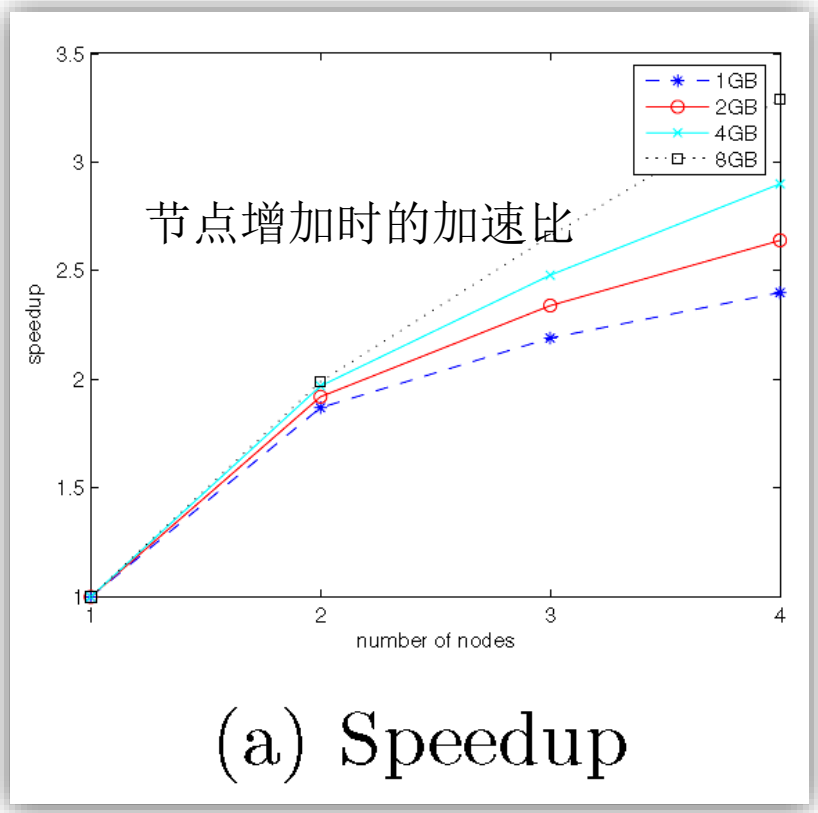
Iteration No	1	
Cluster id	cluster 中心	# of data points assigned
cluster -0	(1, 1)	1
cluster-1	(3.67, 2.67)	3

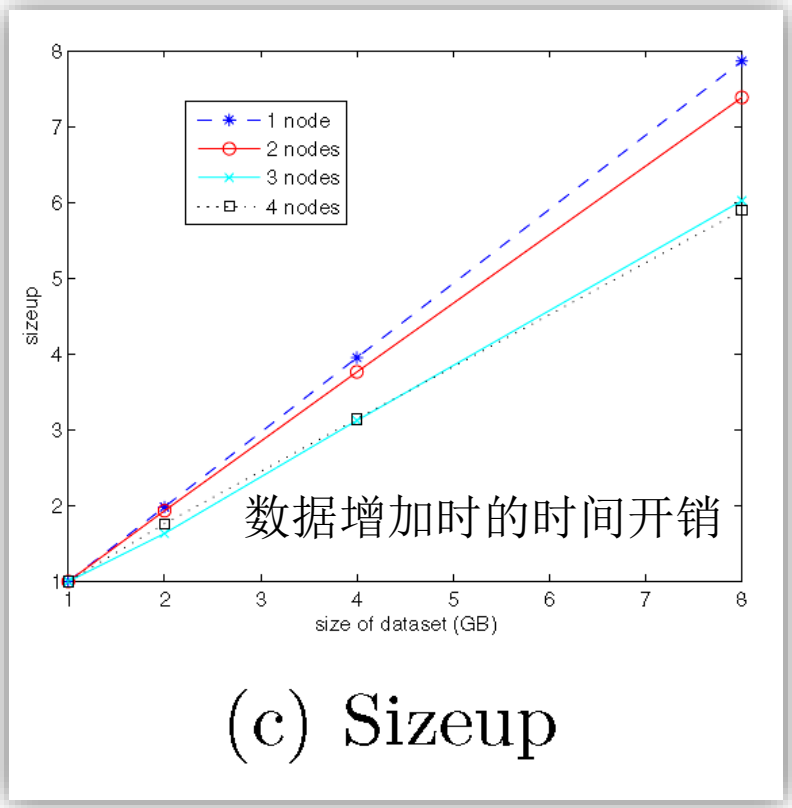
终止迭代

- 在第i次迭代后，已经生成了K个聚类。如果满足了终止条件，即可停止迭代，输出K个聚类

- 终止条件：
  - 设定迭代次数；
  - 均方差的变化（非充分条件）
  - 每个点固定地属于某个聚类
  - 其他设定条件 . . . . . .
  - 与具体的应用高度相关

3. 实验结果与小结





Parallel K-means clustering based on MapReduce
Zhao, Weizhong; Ma, Huifang; He, Qing **Source:** *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 5931 LNCS, p 674-679, 2009, *Cloud Computing - First International Conference, CloudCom 2009, Proceedings*

算法设计实现小结

- 利用MapReduce来并行化K-Means聚类过程是可行的
- 每个节点计算一部分数据的归属，从而实现并行
- 数据间是无关的，但是数据和聚类中心是相关的，因此需要全局文件，但不构成性能瓶颈
- 没有因为并行而降低了算法的精确度（每一个点均保证与每一个cluster center进行了比较）

4. 聚类算法应用实例

NetFlix
百万美元
大奖赛

Netflix公司与大奖赛

- 美国的一家电影在线租赁公司
- 拥有大量用户的影评记录
- 影片推荐系统基于这些影评记录
- 2009年设置大奖赛，能够将推荐正确率提高10%者，将获得100万美元的奖励

Netflix公司与大奖赛

上一个的百万美元大奖的挑战：为那些提供了50个以上评级的观众准确地预测他们的口味，已经被*BellKor's Pragmatic Chaos*团队解决。

下一个百万美元大奖的挑战：为那些*不经常做影片评级或者根本不做评级的顾客*推荐影片，要求使用一些隐藏着观众口味的地理数据和行为数据来进行预测。

竞赛数据

- 训练数据集：由Netflix.com提供的由来自超过480K名随机选择的匿名用户对接近18K部电影的影评（超过100M条）
- 影评的等级范围：从★到★★★★★
- 测试数据集：包含超过2.8M条记录。每条记录包含用户ID、电影ID、日期、影评等级，但是影评信息是空白。测试数据集是训练数据集的一个子集。

竞赛数据

- 有17770个影片的影评文件，每个文件代表一部影片，描述了观众对于该影片的评价

- 评价的指数从★到★★★★★ ，每个影评文件的格式如：
  - 第一行：movieID
  - 其余每行：userID, rating, date

- 两个影评文件中的观众id可能有相同的，也可能不同

对提交算法的要求

- 对于测试数据集中的任一个<用户ID，电影ID>对，算法必须能够预测出该用户对该电影的影评，即给出影评的星级(1-5级)。

算法的评价标准

- 测试集将被划分为不交的两个子集（划分的方法是保密的），对这两个子集中的每一个预测影评，Netflix.com将计算其与对应的真实影评的**均方根误差**(RMSE, Root Mean Squared Error)，计算结果精确到0.0001。

- $\delta = \sqrt{\frac{\sum_{i=1}^{n} d_i^2}{n-1}}$

- 这里n是测量次数，d_i是一组测量值与平均值的误差。

竞赛结果

- Cinematch影片推荐引擎形成了在前述训练数据集上的预测算法，该预测算法对测试数据集所做的预测的RMSE为0.9525。
- 要想拿到百万美金大奖，参赛者至少将预测的精确度提高10%，所以，算法的预测结果的RMSE必须不大于0.8572。
- **BellKor's Pragmatic Chaos** 为本次竞赛的领先团队，他们所提交的算法的RMSE为0.8567。

Netflix中的聚类问题

- 目的：根据观众的评价对这17770部影片进行数据挖掘，输出约400个聚类，使得每个聚类中的影片是相似的

- 考虑：
- 1. 怎样定义及计算这类聚类问题中的相似度
- 2. 怎样表示一个聚类（或聚类中心）
- 3. 对于高维数据怎样预处理
- 4. 怎样减少高维数据的计算量

Ch 9. 数据挖掘基础算法

9.1数据挖掘并行算法研究的重要性

9.2 基于MapReduce的K-Means聚类算法

9.3 基于MapReduce的分类算法

9.4 基于MapReduce的频繁项集挖掘算法

9.3 基于MapReduce的分类并行算法

1. 分类问题的基本描述
2. K-最近邻分类并行化算法
3. 支持向量机SVM分类
4. 朴素贝叶斯分类并行化算法

1. 分类问题基本描述

- 分类(Classification)是机器学习和数据挖掘中最重要、最频繁使用的算法。
- 分类算法的基本作用是：从一组已经带有分类标记的训练样本数据集来预测一个测试样本的分类结果。
- 分类算法的基本描述是：

  一个训练数据集TR = {tr1，tr2, tr3,…}

  每个训练样本tri是一个三元组（tid，A，y）

  其中tid是每个训练样本的标识符，A是一组特征属性值：

  A = {a1，a2，a3,…}， 而y是训练样本已知的分类标记。

tid	age	sex	cm	Kg	发育
1	2	M	80	14	良好
2	3	M	82	12	不良
3	2	F	68	12	良好
…	…	…	…	…	…

分类问题基本描述

- 对于一个测试样本数据集TS = {ts1，ts2, ts3,…}
  每个测试样本ts也是一个三元组（tid，A，y’），其中y’ 未知
- 需要解决的问题是：根据训练数据集来预测出每个ts的未知的分类标记y’
- 训练数据集越大，对测试样本分类标记的预测越准确

训练样本数据

tid	age	sex	cm	Kg	发育
1	2	M	80	14	良好
2	3	M	82	12	不良
3	2	F	68	12	良好
4	2	F	75	17	肥胖
…	…	…	…	…	…

测试样本数据

tid	age	sex	cm	Kg	发育
1	2	F	70	15	?
2	2	M	82	12	?
3	3	F	68	12	?
4	2	M	75	17	?
…	…	…	…	…	…

2. K-最近邻(KNN)分类并行化算法

基本算法设计思想

- K-最近邻是分类器算法中最通俗易懂的一种，计算测试样本到各训练样本的距离，取其中距离最小的K个，并根据这K个训练样本的标记进行投票得到测试样本的标记。
- 加权K-最近邻分类算法的思路是，在根据测试样本的标记进行投票表决时，将根据测试样本与每个训练样本间距离（或相似度）的大小决定训练样本标记的作用大小，基本原则是：距离越近的训练样本其标记的作用权重越大，反之则越小。据此，可以建立一个带加权的投票表决计算模型(比如$y' = \sum Si*yi/\sum Si$, k=[0,k-1],Si为取值0-1的相似度数值，yi为选取出的最邻近训练样本的分类标记值)决定以最终的测试样本的分类标记。
- 算法的思路清晰简单，然而对于海量数据计算量很大，耗费时间较长。

MapReduce并行化算法设计思路

- 基本处理思路是：将测试样本数据分块后分布在不同的节点上进行处理，将训练样本数据文件放在DistributedCache中供每个节点共享访问
- Map阶段对每个读出的测试样本数据ts(trid,A',y')
  - 计算其与每个训练样本数据tr(trid,A,y)之间的相似度S=Sim(A',A) （1：相似度最大，0：相似度最小）
  - 检查S是否比目前的k个S值中最小的大，若是则将(S,y)计入k个最大者
  - 根据所保留的k个S值最大的(S,y)，根据模型y'=∑Si*yi/∑Si计算出ts的分类标记值y'，发射出(tsid,y')
- Reduce阶段直接输出(tsid,y')

MapReduce并行化算法实现

Mapper伪代码

```
class Mapper
 setup(…)
 {
     读取全局训练样本数据文件，转入本地内存的数据表TR中
 }
 map(key, ts)  // ts为一个测试样本
 { Φ → MaxS (k)
   ts  → tsid, A', y'
    for i=0 to TR.lenghth)
    {  TR[i] → trid, A, y
       S = Sim(A, A');
       若S属于k个最大者，  (S, y) → MaxS;
    }
   根据MaxS和带加权投票表决模型计算出y' =∑Si*yi/∑Si
   emit(tsid, y')
 }
```

3. 朴素贝叶斯分类并行化算法

基本问题描述和算法设计思想

- 设每个数据样本用一个n维特征向量来描述n个属性的值，即：X={x1，x2，…，xn}，假定有m个类，分别用Y1, Y2,…，Ym表示
- 给定一个未分类的数据样本X，若朴素贝叶斯分类将未知的样本X分配给类Yi，则一定有P(Yi|X)>P(Yj|X), 1≤j≤m，j≠i
- 根据贝叶斯定理P(Yi|X)=P(X|Yi)*P(Yi)/P(X)，由于P(X)对于所有类为常数，概率P(Yi|X)可转化为概率P(X|Yi)P(Yi)。
- 如果训练数据集中有很多具有相关性的属性，计算P(X|Yi)将非常复杂，为此，通常假设各属性是互相独立的，这样P(X|Yi) 的计算可简化为求P(x1|Yi)，P(x2|Yi)，…，P(xn|Yi)之积；而每个P(xj|Yi)可以从训练数据集近似求得。
- 据此，对一个未知类别的样本X，可以先分别计算出X属于每一个类别Yi的概率P(X|Yi)P(Yi)，然后选择其中概率最大的Yi作为其类别。

MapReduce并行化算法设计思路

- 根据前述的思路，判断一个未标记的测试样本属于哪个类Yi的核心任务成为：根据训练数据集计算Yi出现的频度和所有属性值xj在Yi中出现的频度。
- 据此，并行化算法设计的基本思路是：用MapReduce扫描训练数据集，计算每个分类Yi出现的频度FYi (即P(Yi))、以及每个属性值出现在Yi中的频度FxYij (即 P(xj|Yi))
- 而在MapReduce中对训练数据集进行以上的频度计算时，实际上就是简单地统计Yi和每个xj在Yi中出现的频度
- 在进行分类预测时，对一个未标记的测试样本X，根据其包含的每个具体属性值xj，根据从训练数据集计算出的FxYij进行求积得到FXYi(即P(X|Yi))，再乘以FYi即可得到X在各个Yi中出现的频度P(X|Yi)P(Yi)，取得最大频度的Yi即为X所属的分类。

MapReduce并行化算法实现

训练数据集Yi频度统计Mapper伪代码

```
class Mapper
map(key, tr)  // tr为一个训练样本
{
    tr → trid, A, y
     emit(y, 1)
     for i=0 to A.lenghth)
     {  A[i] → 属性名xni和属性值xvi
        emit(<y, xni, xvi>, 1)
     }
}
```

MapReduce并行化算法实现

训练数据集频度统计Reducer伪代码

```
class Reducer
reduce(key, value_list)  // key 或为分类标记y，或为<y, xni, xvi>
 {
      sum =0
      while(value_list.hasNext())
          sum += value_list.next().get();
      emit(key, sum)
 }
```

- 输出结果为所有Yi的出现频度FYi，以及所有属性值xj在Yi中的出现频度
- 进行未标记测试样本X的分类预测时，可以从这个输出数据文件中快速查找相应的频度值并最终计算出X在各个Yi中出现的频度P(X|Yi)P(Yi)，最后取得最大频度的Yi即为X所属的分类

MapReduce并行化算法实现

测试样本分类预测Mapper伪代码

```
class Mapper
setup(…)
 {   读取从训练数据集得到的频度数据
     分类频度表 FY = { (Yi, 每个Yi的频度FYi) }
     属性频度表 FxY = { (<Yi, xnj, xvj>, 出现频度FxYij ) }
 }
map(key, ts)  // ts为一个测试样本
{   ts → tsid, A
    MaxF = MIN_VALUE; idx = -1;
    for (i=0 to FY.length)
    {  FXYi = 1.0；  Yi = FY[i].Yi; FYi = FY[i].FYi
       for (j=0 to A.length)
       {   xnj = A[j].xnj; xvj = A[j].xvj
           根据<Yi, xnj, xvj>扫描FxY表, 取得FxYij
           FXYi = FXYi * FxYij;
       }
       if(FXYi* FYi >MaxF)  { MaxF = FXYi*FYi;  idx = i; }
    }
    emit(tsid, FY[idx].Yi)
 }
```

4. SVM短文本多分类并行化算法

基本问题描述

本问题是2012年中国第一届“云计算与移动互联网大奖赛”指定的4个大数据并行处理赛题之一，本系研究生组队参加，经过角逐获得1、2、3等奖各一项。

基于MapReduce的查询短文本多分类并行化算法研究。提供了1万条已经标注出所属类别的短文本样本数据作为训练样本，一共有480个类别。测试数据有1000万条查询短文本样本数据，其中有少数不属于这480类的异类测试样本，需要对这些大量的短文本进行分类，并能标识出不属于以上480类的异类样本。

每个短文本样本数据由一个n维的高维特征向量构成

基本算法设计思路

本道题目是高维稀疏空间文本的分类问题。由于大量实践证实SVM 针对高维空间数据训练效果较好，而且分类器的速度较快，因此将使用linear SVM进行处理

- 训练阶段，对于多类（480 类）问题，为了提高分类精度，首先针对每个类做一个2-Class两类分类器；
- 预测阶段，分别用480 个分类器对每个待预测的样本进行分类并打分，选择分类为“是”且打分最高的类别作为该样本可能的预测类别；如打分低于最低阈值，则将该测试样本判定为不属于480类的异类

基本算法设计思路

- 为了提升训练和分类速度，上述所有算法都在MapReduce框架下实现，分两步MapReduce处理完成

  第一步：用训练数据产生480个2-class分类器模型

  Map：将每个训练样本的分类标签ClassID作为主键，输出(ClassID, <true/false，特征向量>)

  Reduce: 具有相同ClassID的键值对进入同一个Reduce，然后训练出一个2-Class SVM分类模型

  共输出480个2-Class SVM分类模型

大规模短文本多分类并行化算法

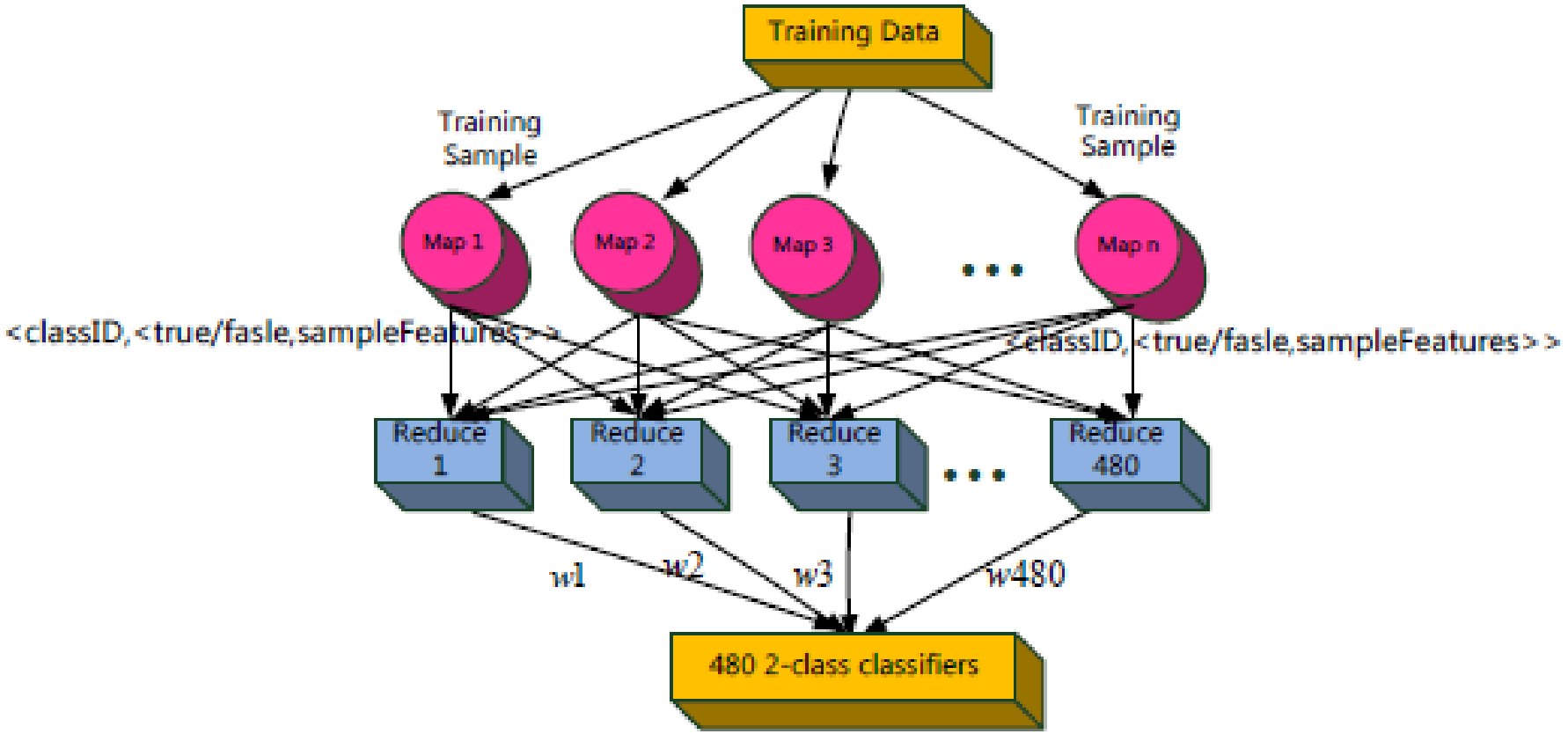


图 1.基于 MapReduce 的 480 个两类分类器并行训练算法

基本算法设计思路

- 为了提升训练和分类速度，上述所有算法都在MapReduce框架下实现，分两步MapReduce处理完成

  第二步：用480个2-Class分类器模型处理测试数据

  Map：将每个测试样本，以SampleID作为主键，输出(SampleID, <LableID，评分Score>)

  Reduce: 具有相同SampleID的键值对进入同一个Reduce，然后以最高评分者对应的标记作为该样本最终的标记；虽然是最高评分，但仍然低于最小阈值，则判定为不属于已知的480个类

大规模短文本多分类并行化算法

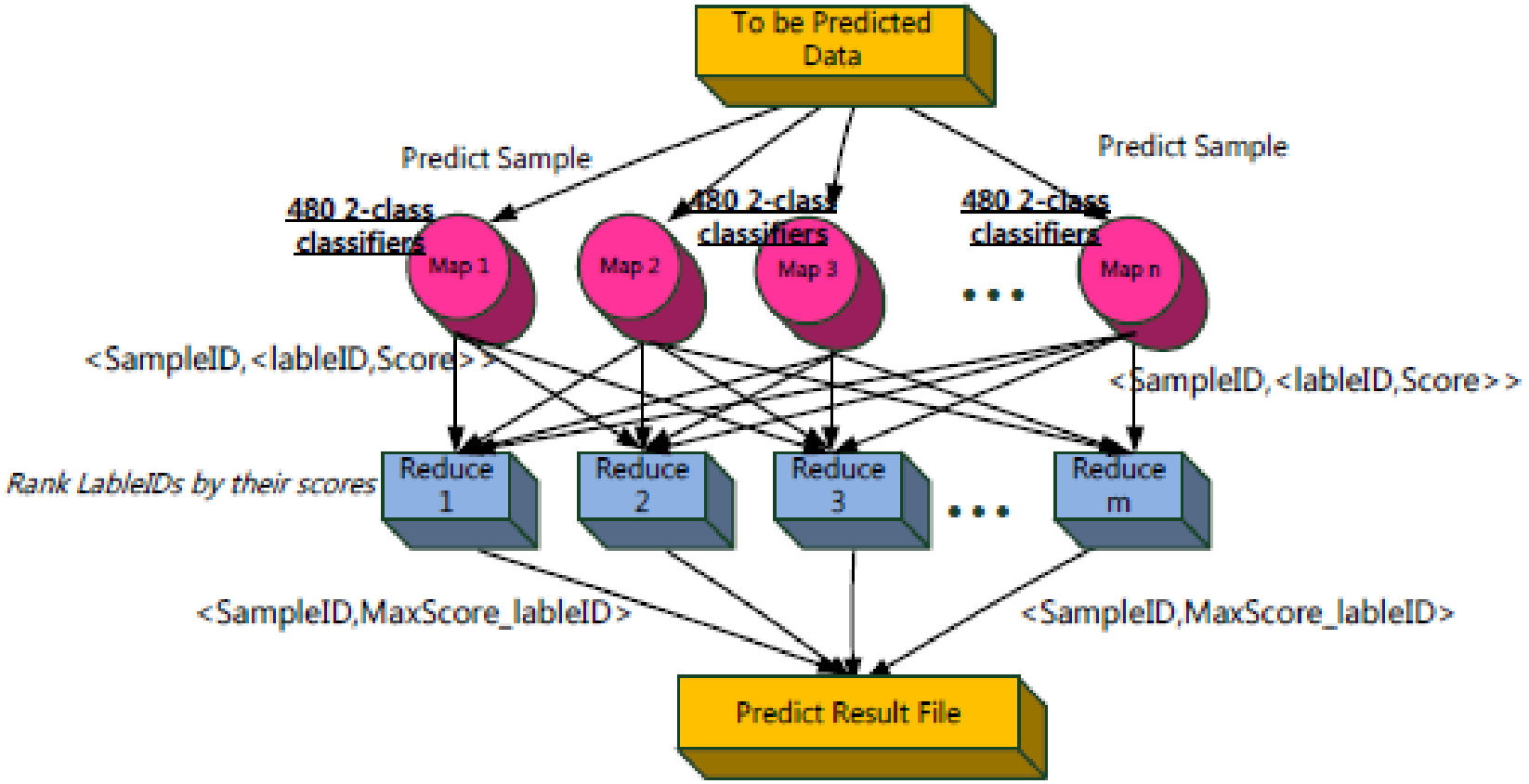


图 2. 基于 MapReduce 的 480 个两类分类器并行预测算法

Ch 9. 数据挖掘基础算法

9.1数据挖掘并行算法研究的重要性

9.2 基于MapReduce的K-Means聚类算法

9.3 基于MapReduce的分类算法

9.4 基于MapReduce的频繁项集挖掘算法

9.4 基于MapReduce的频繁项集挖掘算法

1.频繁项集挖掘问题概述

2. 现有算法概述

3. PSON：基于MapReduce的并行化算法

4. 并行化算法实验结果

1. 频繁项集挖掘问题概述

本研究组进行了基于MapReduce的频繁项集挖掘算法研究

PSON: A Parallelized SON Algorithm with MapReduce for Mining Frequent Sets

Tao Xiao, Shuai Wang, Chunfeng Yuan, Yihua Huang

The Fourth International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2011)， Tianjin，Dec. 9-11, 2011

基本设计实现思路是：

根据基本的Apriori 算法和SON算法，研究实现并行化的频繁项集挖掘算法

What is transaction and itemsets ?

- A transaction is composed of an id and a set of items

TID	Items
T100	I1, I2, I5
T200	I2, I3, I4
T300	I3, I4
T400	I1, I2, I3, I4

- There are 4 transactions in the figure above
- The first transaction (T100) has 3 items, {I1, I2, I5} is an itemset
- The length of {I1, I2, I5} is 3, so it is called a 3-itemsets

- An itemset, whose length is *k*, is referred as a *k*-itemset

What is transaction and itemsets ?

- Suppose *I* is an itemset consisting of items from the transaction database *D*
  - Let *N* be the number of transactions *D*
  - Let *M* be the number of transactions that contain all the items of *I*
  - *M / N* is referred to as the *support* of *I* in *D*

TID	Items
T100	I1, I2, I5
T200	I2, I3, I4
T300	I3, I4
T400	I1, I2, I3, I4

Example

Here, N = 4, let *I* = {I1, I2}, than M = 2

because *I* = {I1, I2} is contained in transactions T100 and T400

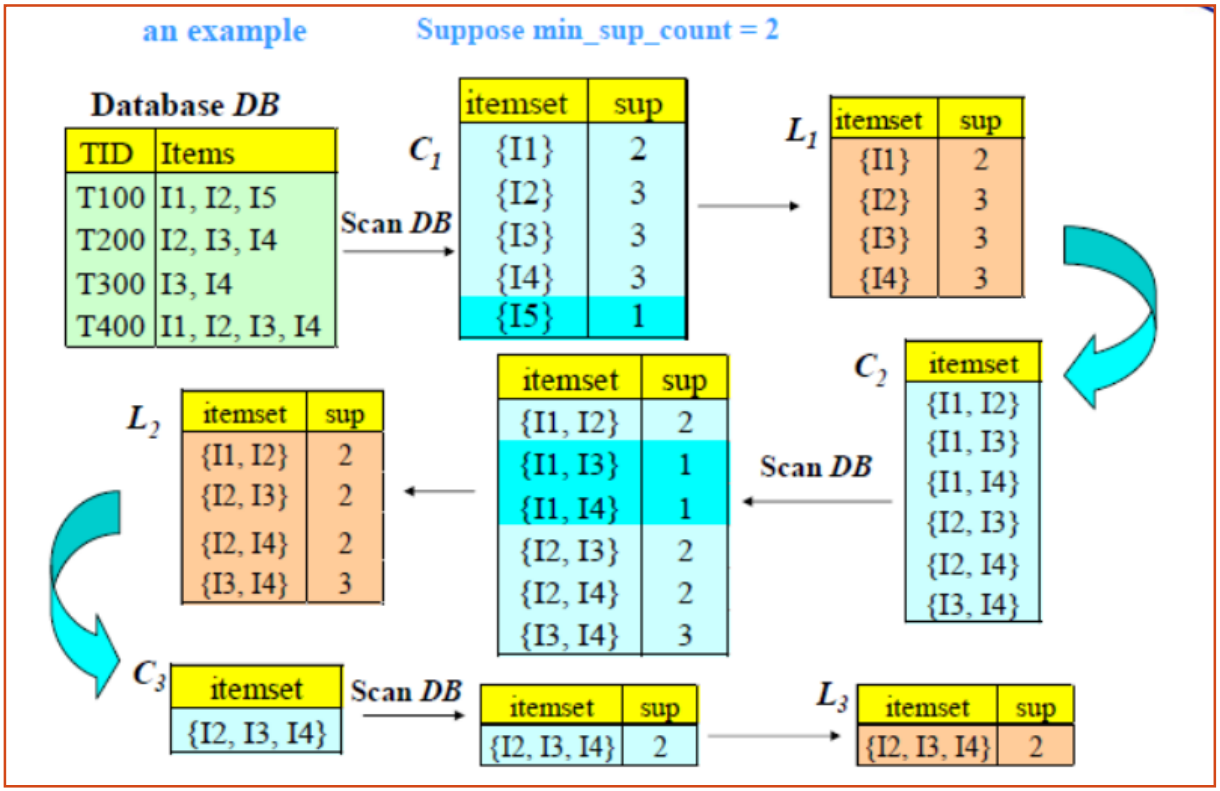
so the support of *I* is 0.5 (2/4 = 0.5)

- If *sup*(*I*) is no less that an user-defined threshold, then *I* is referred to as a frequent itemset
- Goal of frequent sets mining
  - To find all frequent *k*-itemsets from a transaction database (*k* = 1, 2, 3, ...)
- 枚举计算的时间复杂度是：$O(2^n*N*t)$， n是Item的总数，N是Transaction总数，t是每个Transaction平均包含的Item数

2. 现有算法概述

Apriori算法

- **Basic Idea**
  - A classic frequent sets mining algorithm
  - Needs multiple passes over the database
  - In the first pass, all frequent 1-itemsets are discovered
  - In each subsequent pass, frequent (k+1)-itemsets are discovered, with the frequent k-itemsets found in the previous pass as the seed (referred to as *candidate itemsets*)
  - Repeat until no more frequent itemsets can be found



* R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," in proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, August 29-September 1, 1994

SON算法

- Basic idea
  - Divide the whole database into several non-overlapping partitions
  - For each partition, discover all the frequent itemsets (referred to as *local frequent itemsets*)
  - Merge all the local frequent itemsets from all the partitions (referred to as *global candidate itemsets*)
  - Remove those that are not actually frequent in the whole database, generating *global frequent itemsets*
- Lemma
  - An itemset that is not local frequent in any of the partitions cannot be global frequent（不是局部频繁的一定不是全局频繁的）
  - A global frequent itemset must appear as local frequent in at least one of the partitions（全局频繁的一定是局部频繁的）

* A. Savasere, E. Omiecinski, and S. Navathe, “An efficient algorithm for mining association rules in large databases,” in proceedings of the 21st VLDB Conference Zurich, Swizerland, 1995

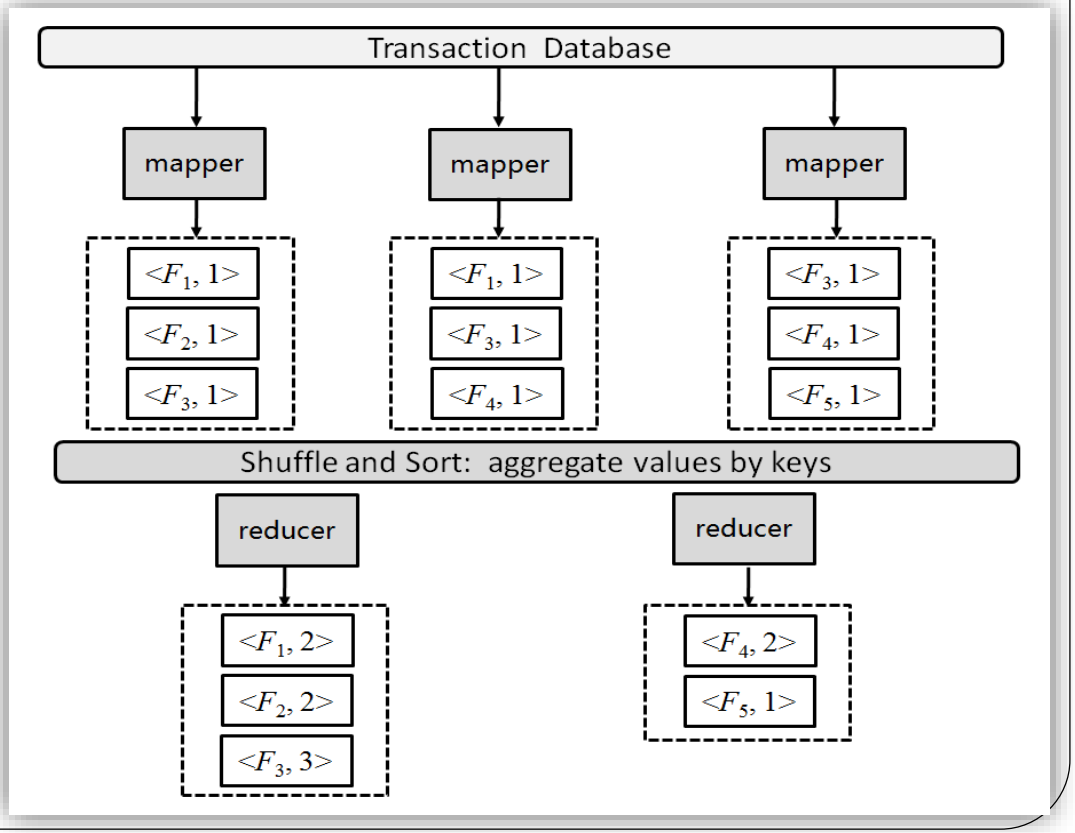
3. PSON：基于MapReduce的并行化算法

- Motivation to Parallelize SON Based on MapReduce
  - Processing one partition doesn't need any information from any other partition
  - Each partition can be processed concurrently
  - SON is naturally suitable for parallelization
- Preparing data
  - Store the transaction database into DFS
  - The whole database will be automatically divided into several non-overlapping chunks
  - Chunks correspond to the partitions in SON
- Map tasks
  - Each chunk is processed by one mapper node to find local frequent itemsets for that chunk
- Reduce tasks
  - Local frequent itemsets of the same length are processed by one reduce node
  - Each node counts for each global candidate itemset it receives
  - Then decides which are global frequent itemsets
- Run two MapReduce jobs to generate all frequent itemsets
  - 1st job: **generate all global candidate itemsets**
  - 2nd job**: identify global frequent itemsets from global candidate itemsets**

PSON：基于MapReduce的并行化SON算法

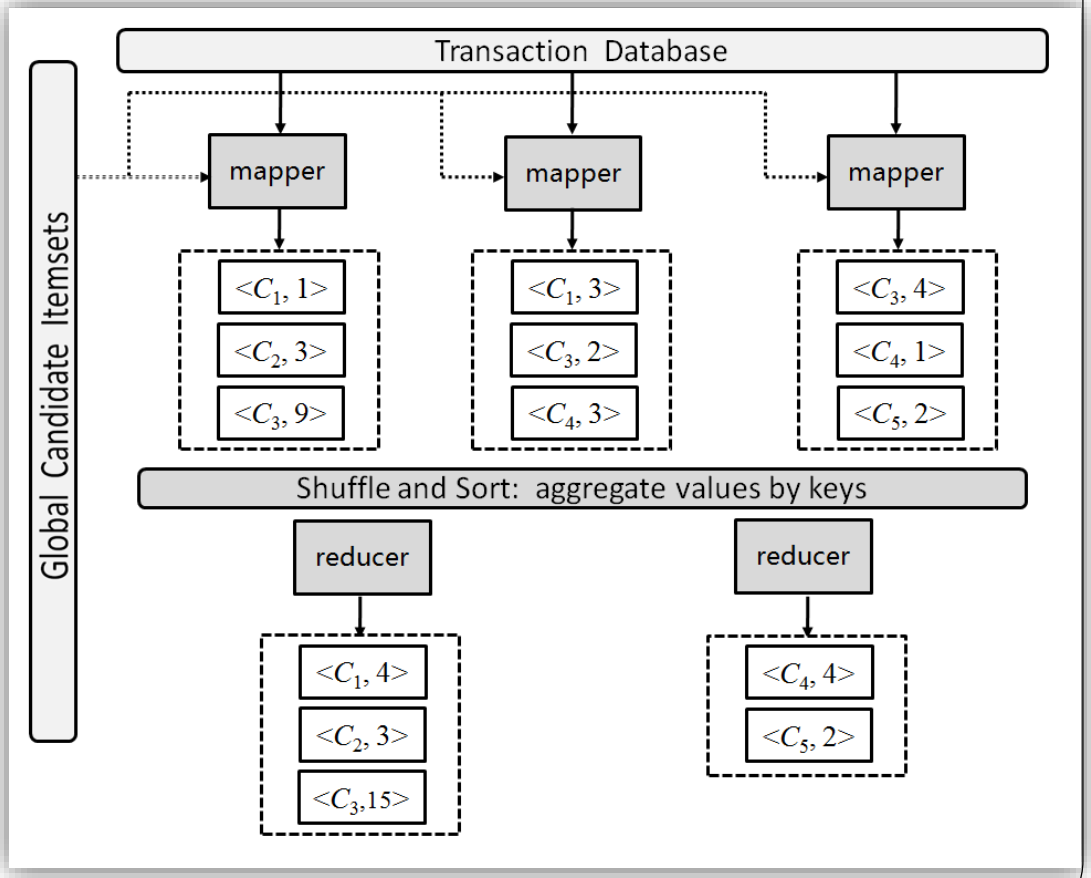
1st MapReduce Job: generate all global candidate itemsets

- **Map phase**
  - Each map node takes in one partition and generates local frequent itemsets for that partition using Apriori algorithm.
  - For each local frequent itemset *F*, emits key-value pair <*F*, 1>. Here, the value 1 is only to indicate that *F* is a local frequent itemset for that partition.
- **Shuffle and Sort phase**
  - The same local frequent itemsets are sent to one reduce node.
- **Reduce phase**
  - Each reduce node emits one and only one key-value pair <F, 1> to DFS
- **Finally**
  - Merging all the pairs in DFS gives us all global candidate itemsets



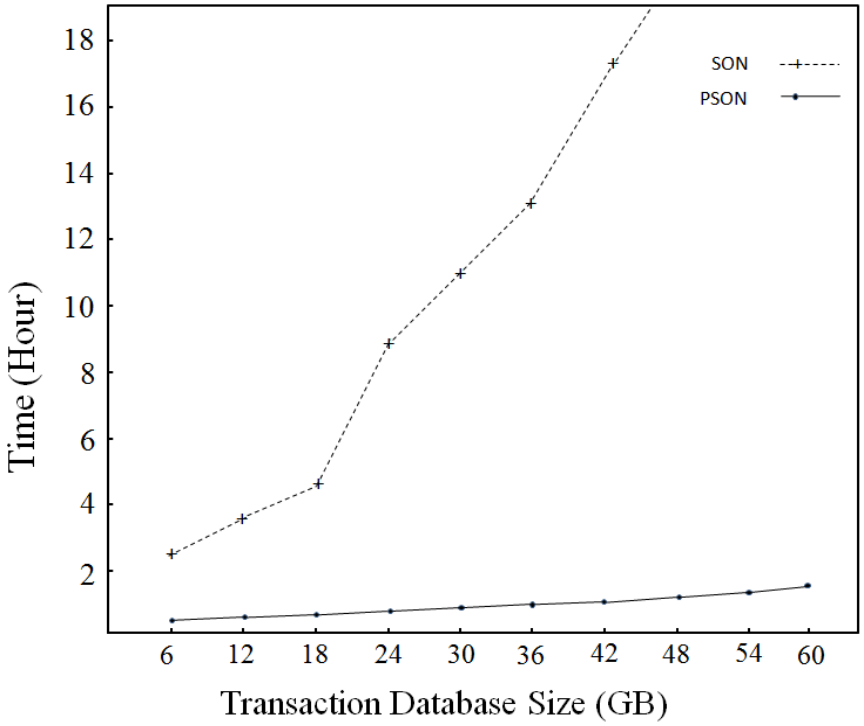
2nd MapReduce Job: identify global frequent itemsets

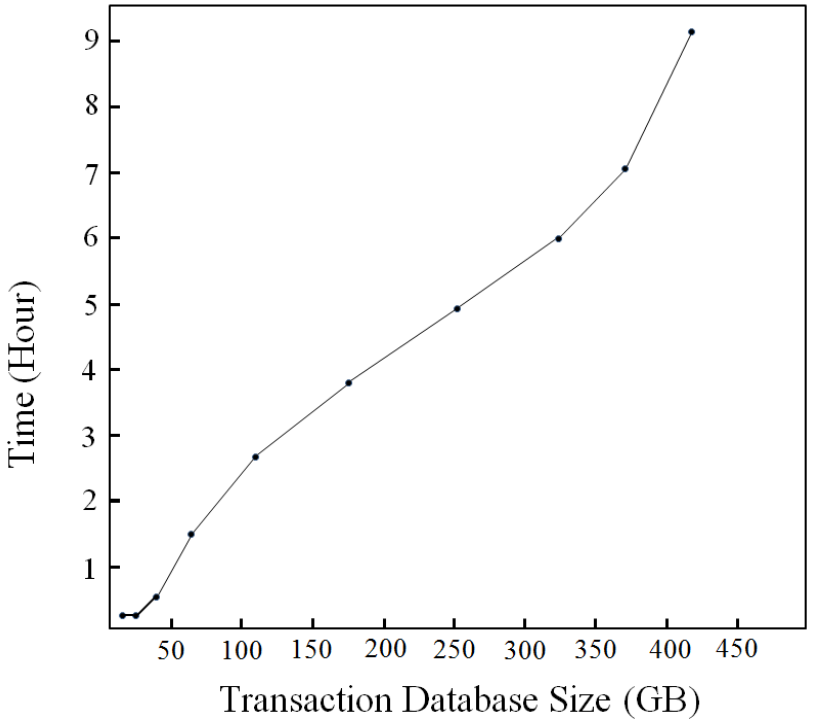
- **Assumption**
  - Each node is given a full duplicate of the global candidate itemsets generated by the 1st MapReduce job beforehand
- **Map phase**
  - Each map node counts for each of the global candidate itemsets in the partition the map node is assigned
  - Then emits pairs like <*C*, *v*> where *C* is a global candidate itemset and *v* is the count of it in that partition
- **Shuffle and Sort phase**
  - Each global candidate itemset and its counts in all the partitions are sent to one reduce node
- **Reduce phase**
  - For each global candidate itemset *C*, reduce node adds up all the associative counts for *C* and emits only the actual global frequent itemsets to DFS



4. 并行化算法实验结果

- The transaction database size varies from 6GB to 60GB，with the number of transactions varies from 1 million to 500 billion

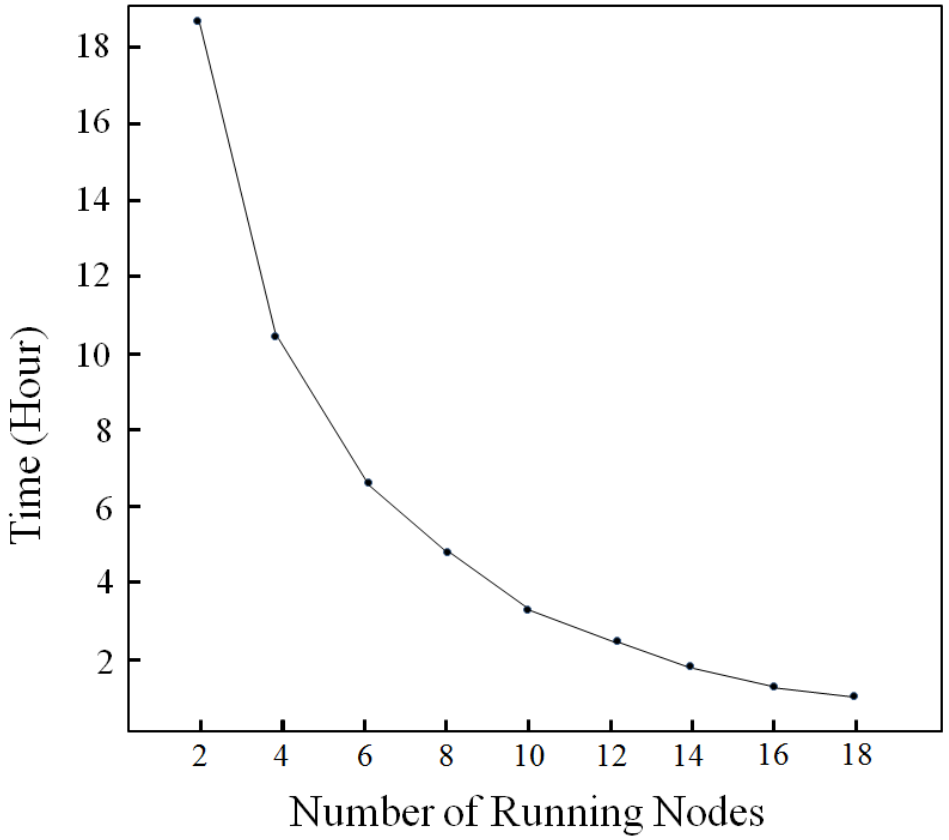




- Conclusion: When the size of the database reaches a threshold of hundreds of GB, PSON can finish running in an acceptable period of time，achieving a good performance in scale-up

并行化SON算法实验结果

- Number of running nodes varies from 2 to 18



- Conclusion: PSON can achieve a good performance in speed-up

Thank You !