MR-GSpar：一种基于MapReduce的大图稀疏化算法

陈德华[1] 周 蒙[1] 孙延青[2] 郑亮亮[1]
(东华大学计算机科学与技术学院 上海 201620)[1] (上海市浦东新区电子政务管理中心 上海 200135)[2]

摘 要 图的稀疏化是图聚类分析中数据预处理的关键操作，已得到广泛的关注。针对图数据日益普及、规模不断增大的现状，提出了一种基于MapReduce的面向大规模图的稀疏化算法，即MR-GSpar算法。该算法在MapReduce并行计算框架的基础上，通过对传统的最小哈希(Minhash)算法的并行化改造，使其可在分布式的集群环境中实现对大规模图数据的高效稀疏化处理。真实数据集上的实验表明了该算法的可行性与有效性。

关键词 图稀疏化，Minhash，MapReduce框架，MR-GSpar算法
中图法分类号 TP3111 **文献标识码** A

MR-GSpar: A Distributed Large Graph Sparsification Algorithm Based on MapReduce

CHEN De-hua[1] ZHOU Meng[1] SUN Yan-qing[2] ZHENG Liang-liang[1]
(School of Computer Science & Technology, Donghua University, Shanghai 201620, China)[1]
(E-Government Management Center, Pudong District, Shanghai 200135, China)[2]

Abstract As an important data pre-processing operation, graph sparsification has attracted wide attentions from the area of database. Nowadays the graph data is becoming popular and scale. Thus this paper proposed an efficient parallel graph sparsification algorithm, namely the MR-GSpar algorithm. The MR-GSpar algorithm is presented by reforming the traditional Minhash algorithm into a parallel and distributed algorithm using MapReduce framework, which can archive efficient sparsification on large-scale graph data in a large machine cluster environment. Experiments on real datasets show that the algorithm is feasible and effective.

Keywords Graph sparsification, Minhsh, MapReduce framework, MR-GSpar algorithm

1 引言

许多现实世界中的复杂网络如蛋白质交互网络、社会网络、传感器和通讯网络、交通运能网络等都可以自然地建模为图结构。在这类图中，结点表示网络中的实体，边则表示实体间的联系。例如，社会网络就可以被表示为一个无向图，其中结点代表参与社会网络的个体，边代表个体与个体之间的朋友关系。近年来，随着计算机信息技术的推广普及，尤其是当前Web2.0网络如人人网、腾讯微博、FaceBook、Twitter等应用的迅猛发展，图数据的规模普遍呈现出指数增长趋势，产生了许多各式各样的海量图[1]。

图聚类[2]是图数据分析与挖掘应用的基本功能之一，其目的是对图中的结点进行划分从而形成不同的聚簇，并且使得同一个聚簇中的结点之间的关联度比较紧密，簇与簇之间各个结点则关联度比较低。图聚类已在社会网络的社区发现、蛋白质复合物检测等实际应用中得到了广泛的运用。那么，面对日益出现的大规模图数据，如何对其进行高效的聚类分析，以挖掘出有效的潜在信息，便成为图挖掘研究领域的核心问题之一。

数据抽样[3]是提高数据挖掘算法执行效率的一种途径。其基本思路是通过从整个数据集抽取出部分样本数据，然后在抽样数据上执行数据挖掘操作，从而在较短时间内获得近似的挖掘结果。因此，在图聚类应用中，通过对图数据进行点或边抽样，即经过图的稀疏化处理得到稀疏图，然后在稀疏化之后的图上进行图聚类，这已成为提高图聚类分析效率的有效方法。

图的稀疏化[3]作为图聚类分析中数据预处理的关键操作，已得到了广泛的关注。目前，已有许多不同图稀疏化方法被提出，包括：k-最近邻图、Power Method以及L-spar等。但是，这些已有的图稀疏化方法均是针对规模较小的图数据，且只适合在单机环境中运行。

面对规模日益庞大的图数据，单个工作站不管是在CPU计算能力还是在内存消耗上均无法满足需求，从而导致图稀疏化处理无法正常执行。MapReduce作为一种并行编程模型[4,5]，可实现上百乃至上千台计算机的互联，能将巨大的系统资源池连接在一起，形成庞大的机器集群，特别适用于大规模数据的并行处理。因此，本文旨在研究一种基于MapReduce的面向大图数据的稀疏化算法。

Minhash算法作为一种位置敏感哈希算法(Locality Sensitive Hashing，简称LSH)，用于快速估算两个集合的相似度，已广泛应用于文本数据、多媒体数据的近似搜索中。具体而言，Minhash算法是一种基于Jaccard相似度[6,7]的近似计

到稿日期：2013-01-03 返修日期：2013-04-28 本文受国家自然科学基金项目(61070031，61070032)资助。
陈德华 男，副教授，硕士生导师，主要研究方向为数据仓库、大数据分析、图数据挖掘；周 蒙(1989—)，女，硕士生，主要研究方向为大数据分析、数据挖掘，E-mail：kibumzm@163.com。

算方法，其大致思想是使用 K 个哈希函数分别对两个集合 A 和 B 求哈希值，这样每个集合都将得到 K 个最小哈希值。那么，集合 A、B 的相似度则为最小哈希值相同的元素个数与总的元素个数之间的比例。近年来，有研究者[3]在对图聚类的特征进行分析之后，得出一种简单的启发式图聚类规则，即同一聚簇内各结点具有非常相似的邻居节点集。换言之，具有相似邻居结点集的两个结点极有可能聚到一类中。此启发式规则在图的稀疏化中，意味着由这两个结点组成的连接边应被保留下来。反之，如果两个结点的邻居结点集不大相似，则由这两个结点组成的连接边则可被舍弃。这与 Minhash 算法的思想相贴合。因此，文献[3]尝试利用 Minhash 来实现一种图的局部稀疏化算法 L-spar，该算法针对图中每个结点的所有邻接边，按照权重排序，只保留那些权重值大于阈值的边，删除其余的边，达到图的局部稀疏化目的。

正是基于上述考虑，本文以 MapReduce 框架为基础，探讨在分布式环境下如何实现大规模图数据的稀疏化处理。为此，本文将 MapReduce 并行框架与 Minhash 算法相结合，提出了一种基于 MapReduce 的分布式图稀疏算法，即 MR-GSpar。具体而言，MR-GSpar 算法利用 MapReduce 的并行计算模型，实现图稀疏化过程中 4 个子任务的快速计算：1）邻居结点集的计算；2）结点的 Minhash 签名计算；3）结点的签名 hash 存储；4）图稀疏化处理。同时本文在开源 MapReduce 开发工具 Hadoop 的支撑下，实现了 MR-GSpar 算法，并对其性能进行了实验。实验结果表明，MR-GSpar 算法对大图数据的稀疏化处理具有良好的性能。

2 背景知识

本节给出相关技术的背景知识，主要围绕 Minhash 和 MapReduce 两种技术进行阐述。

2.1 最小哈希算法 Minhash

如前所述，Minhash 算法是一种基于 Jaccard 系数的集合相似度计算方法。其中，Jaccard 系数又称为 Jaccard 相似度[7]，它是度量两个集合相似程度的测度值。给定两个非空集合 A 和 B，A 与 B 的 Jaccard 系数可通过式(1)计算得到：

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|} \tag{1}$$

从式(1)可见，两集合的 Jaccard 系数为两集合交集与并集之间的比值。显然，A 和 B 越相似，$J(A,B)$就越大。

在集合元素数目庞大的情况下，通过计算交集和并集，进而得到两集合之间的 Jaccard 系数，此方法效率很低。

Minhash 算法基于 Jaccard 系数的原理，首先利用哈希函数 h 分别计算集合 A 和 B 的所有元素，再选取出两个集合的最小哈希值 Minhash(A)和 Minhash(B)，则 Minhash(A)和 Minhash(B)相等的概率等于两个集合的相似度，如式(2)所示。

$$\Pr[mhash(A)=mhash(B)]=\frac{|A\cap B|}{|A\cup B|} \tag{2}$$

因此，在 Minhash 算法中，两个集合的相似度计算问题则转化为计算两个集合最小哈希值相等的概率问题，大大提高了集合相似性计算效率。

2.2 MapReduce

MapReduce 是由 Google 公司首先提出的一种分布式计算架构，用于海量数据集（TB 级以上）的并行运算。MapReduce 使得用户在进行并行计算开发时，只需关注与具体应用相关的 Map 和 Reduce 的处理逻辑本身，而将其余复杂的并行事务交给底层系统去处理。

MapReduce 主要来源于函数式和矢量两种编程思想，其计算过程如图 1 所示。

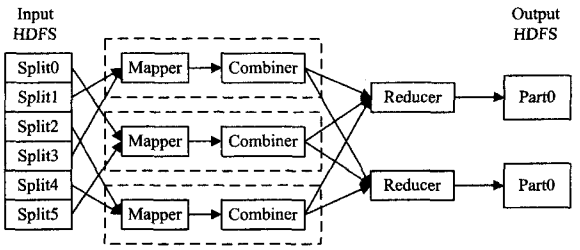

图 1 MapReduce 计算过程示意图

由图 1 可以看出，一个 MapReduce 任务一般会经过以下几个阶段的处理：

1）Mapping 阶段：在此阶段中，每个 Map 函数接受若干独立的数据单元（split），并产生中间结果的键值对〈*key*，*value*〉集合。

2）Combine 阶段：对 Map 操作所输出的中间结果，依据 *key* 进行排序、分组。

3）Reducing 阶段：该阶段中，遍历所有中间数据，并对每一个唯一的 *key* 值执行用户自定义的 Reduce 函数，得到的键值对〈*key*，*value*〉作为结果输出。

Hadoop 是一种开源 MapReduce 编程工具，已被广泛用于开发 MapReduce 的应用。本文实验中也是使用 Hadoop 来实现 MR-GSpar 算法。运行于 Hadoop 平台下的 MapReduce 应用程序由一个 Mapper 类、Reducer 类和一个创建 JobConf 的驱动函数组成，需要时还可以包括一个 Combiner 类，这个类实际上也是 Reducer 类的一种实现。

3 问题阐述

如引言所述，L-Spar 算法是新近提出的一种基于 Minhash 的局部图稀疏化算法。该算法的主要思想是：对于图中的每条边 $v(i,j)$，根据组成边的两结点 i 和 j 之间的 Jaccard 相似度来决定它是保留还是舍弃。其中，结点 i 和结点 j 的 Jaccard 相似度由式(3)计算得到：

$$\mathrm{Sim}(i,j)=\frac{|Adj(i)\cap Adj(j)|}{|Adj(i)\cup Adj(j)|} \tag{3}$$

式中，$Adj(i)$表示与结点 i 相邻的结点集，$Adj(j)$表示与结点 j 相邻的结点集。

为了高效地计算得到 Sim(i，j)，L-spar 算法借用了 Minhash 算法在集合相似度计算的优势。L-spar 算法的具体描述如图 2 所示。

输入：图 G〈V，E〉和稀疏化系数 e

输出：稀疏图 G′(V′，E′)，其中 V′与原始图 G 的 V 相等，E′是 E 的子集

1）对 G 中的每个结点 $v_i\in v$，分别计算得到该结点的度 d_{v_i} 和边集 E_{v_i}；

2）对 E_{v_i} 中的每一条边〈v_i，v_j〉，分别计算得到 v_i 的邻居结点集 Adj(v_i)和 v_j 的邻居结点集 Adj(v_j)；

3）根据 Adj(v_i)和 Adj(v_j)，利用 Minhash 算法计算得到邻接边〈v_i，v_j〉的 minhash(v_i，v_j)，则〈v_i，v_j〉的相似度等于 minhash(v_i，v_j)；

4）对 v_i 的所有邻接边 E_{v_i}，根据其相似度大小进行排序；

5）从 E_{v_i} 中选取相似度排名前 d_i^e 条的边作为稀疏图 G′的边。

图 2 L-Spar 算法

L-Spar 算法是为单机环境下的图稀疏化处理而设计的，

只适用于规模不大的图数据。面对大规模的图数据，MapReduce 计算框架是一种有效的并行处理方案。为此，本文将 MapReduce 引入到图稀疏化中，对已有的 L-Spar 算法进行并行化改造，使其实现高效并行的大规模图数据稀疏化处理，从而提出一种全新的基于 MapReduce 的图稀疏化算法即 MR-GSpar 算法。

4 MR-GSpar 算法

本文提出的 MR-GSpar 算法主要由 4 个步骤组成，分别为：1)邻居结点集的计算；2)结点的 Minhash 签名计算；3)结点的签名 hash 存储；4)图稀疏化处理。各个步骤的具体设计思路将在下面各小节中进行阐述。

4.1 邻居结点集的计算

MR-GSpar 算法的步骤 1)是通过一组 Map 任务计算图中各个结点的邻居结点集，具体处理过程如图 3 所示。

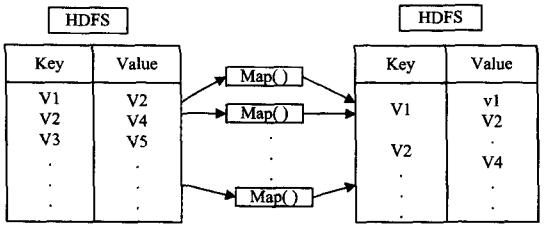

图 3 邻居结点集的计算过程

此步骤中，Map 任务接受一组〈$key=v_i$，$value=v_j$〉的输入，其中 v_i，v_j 表示的是图中相邻的结点。经过 Map 的计算之后，可以得到〈$key=v_i$，$value=list[N_i]$〉对的输出，其中 $list[N_i]$表示结点 v_i 的邻居结点集。最后将结果输出到 HDFS 上。

此步骤的 Map 任务用形式化描述如下：

Map：

〈$key_1=v_i$，$value_1=list[v_i]$〉

→〈$key_2=v_i$，$value_2=list[N_i]$〉

4.2 结点的 Minhash 签名计算

MR-GSpar 算法的步骤 2)是计算图中各个结点的 Minhash 签名。为此，MR-GSpar 算法通过一组 Map 任务和 Reduce 任务的组合来计算图中各个结点的签名 Sig_{v_i}，具体处理过程如图 4 所示。

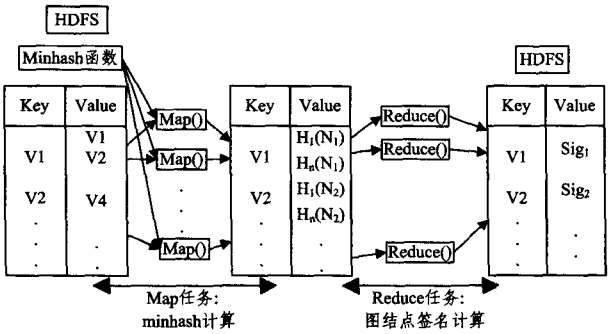

图 4 结点 Minhash 签名的计算过程

从图 4 可以看出，此步骤中的 Map 任务的输入为步骤 1)的输出，即结点的邻居结点集〈$key=v_i$，$value=list[N_i]$〉。同时，Map 任务也接受 k 个 Minhash 函数作为输入。经过哈希计算之后，得到结果〈$key=v_i$，$value=H_m(N_i)$〉($m=1,2,\cdots,k$)，其中 $H_m(N_i)$表示每个结点的 k 个 Minhash 值列表。该输出作为 Reduce 任务的输入，经过 Reduce 计算得到结果〈$key=v_i$，$value=Sig[i][m]$〉，其中 $Sig[i]$为结点 v_i 的签名序列，表示形式是一个二元组。最后将输出的结果 $Sig[i][m]$以二元组形式存储到 HDFS 上。

此步骤 Map 任务和 Reduce 任务用形式化描述如下：

Map：

〈$key_1=v_i$，$value_1=list[N_i]$〉

→〈$key_2=v_i$，$value_2=list[H_m(N_i)]$〉

($m=1,2,\cdots,k$)

Reduce：

〈$key_2=v_i$，$value_2=list[H_m(N_i)]$〉

→〈$key_3=v_i$，$value_3=Sig[i][m]$〉

结点的 Minhash 签名计算步骤分为如下两步执行。

(1)Map 任务详细过程如下所示：

输入：〈v_i，$list[N_i]$〉，其中 key=v_i 为图中的结点，value=$list[N_i]$为结点的邻居结点集；k 个不同的 Minhash 函数。

输出：〈v_i，$list[H_m(N_i)]$〉，其中 value=$list[H_m(N_i)]$为结点的 Minhash 值列表。

```
1. list[Hm]←ϕ/ * 初始化结点的邻居结点集的 Minhash 值列表 * /
2. foreach vi in Graph do
     for m in k
     / * 对结点的邻居结点集进行 k 次 Minhash，并将计算出来的
       hash 值存放到列表 Hm 中 * /
     Hm=Minhash(list[Ni])
     end for
   end foreach
```

(2)Reduce 任务过程主要是利用 Map 阶段的结果，计算结点的签名矩阵，具体过程如下所示：

输入：〈v_i，$list[H_m(N_i)]$〉，结点的 Minhash 值列表。

输出：〈v_i，Sig[i][m]〉，即图中结点的签名矩阵。

```
1. Sig[i][m]←ϕ / * 初始化结点的签名矩阵 * /
2. foreach vi in Graph do
     Sig[i]=sortSig(Hm)
/ * 将结点的 hash 值列表排序，依次存入结点的签名矩阵中 * /
end foreach
```

4.3 结点签名的 hash 存储

MR-GSpar 算法的步骤 3)是针对图中的每个结点，计算得到其邻接边是否属于该结点所在的稀疏子图。MR-GSpar 算法通过一组 Map 任务和 Reduce 任务的组合来计算图中各个结点 hash 后匹配的 hash 桶的数目，具体处理过程如图 5 所示。

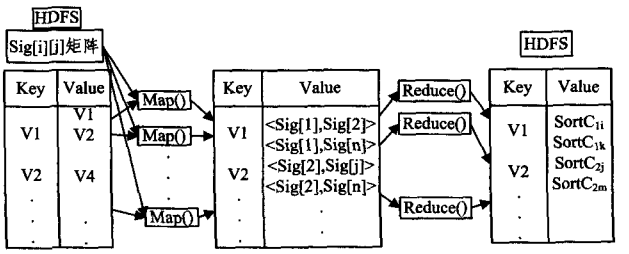

图 5 结点签名的 hash 存储计算过程

此步骤将 4.1 节计算的结点的邻居结点集的〈$key=v_i$，$value=list[N_i]$〉作为 Map 阶段的输入，同时将结点的签名矩阵 $Sig[i][m]$也作为输入，计算得出签名集〈$key=v_i$，$value=list[S(Sig[i],Sig[j])]$〉，其中 $list[S(Sig[i],Sig[j])]$表示结点 v_i 和 v_j 的签名列表集。该输出作为 Reduce 阶段的输入，hash 函数也作为输入，hash 函数的作用是根据 $Sig[i]$计

算结点的桶号，最后将〈$key=v_i$，$value=list[SortC_{ij}]$〉作为输出。其中 $list[SortC_{ij}]$表示的是经过排序的结点 v_i 和 v_j 的签名列表通过 hash 函数计算之后匹配的桶的数目列表。

此步骤的 Map 任务用形式化描述如下：

Map：

〈$key_1=v_i$，$value_1=list[N_i]$〉

→〈$key_2=v_i$，$value_2=list[S(Sig[i],Sig[j])]$〉

Reduce：

〈$key_2=v_i$，$value_2=list[S(Sig[i],Sig[j])]$〉

→〈$key_3=v_i$，$value_3=list[SortC_{ij}]$〉

（j∈邻居结点集）

结点签名的 hash 存储计算步骤分为如下两步执行。

（1）Map 任务详细过程如下所示：

输入：〈v_i，$list[N_i]$〉，其中 key=v_i 为图中的结点，value=$list[N_i]$为结点的邻居结点集，结点的签名矩阵 Sig[i][m]。

输出：〈v_i，$list[S(Sig[i],Sig[j])]$〉，其中 value=list[S(Sig[i]，Sig[j])]表示结点邻接边的签名集。

```
1. list[S]←ϕ/* 初始化结点的邻接边的签名集列表 */
2. foreach v_i in Graph do
     for v_j in list[N_i]
     /* 分别找出对应于结点和邻居结点集中的结点的签名序列 */
     temp1=FindSignature(v_i,Sig)
     temp2=FindSignature(v_j,Sig)
     /* 函数 FindSignature(x,Sig)返回在签名矩阵 Sig 中 x 结点
        的签名序列 */
     S(Sig[i],Sig[j])=Integration(temp1,temp2)
     /* 函数 Integration(x,y)返回 x 和 y 组合的集合 */
     end for
   end foreach
```

（2）Reduce 任务过程主要计算结点和其邻居结点签名的 hash 匹配桶的数目，并将该匹配数目排序。详细过程如下所示：

输入：〈v_i，$list[S(Sig[i],Sig[j])]$〉，其中 key=v_i 为图中的结点，value=$list[S(Sig[i],Sig[j])]$表示结点邻接边的签名集。

输出：〈v_i，$list[SortC_{ij}]$〉，其中 value=$list[SortC_{ij}]$表示经过排序之后的结点与邻接结点匹配的数目。

```
1. list[SortC_ij]←ϕ/* 初始化排序后的结点和邻居结点签名匹配的
   桶数列表 */
2. foreach v_i in Graph do
     foreach 〈Sig[i],Sig[j]〉 in list[S]
     /* 分别对结点和邻居结点的签名进行 hash,记录下桶号 */
     hashtable1=Minhash(Sig[i])
     hashtable2=Minhash(Sig[j])
Cout_ij=MatchTable(hashtable1,hashtable2)
/* 函数 MatchTable(x,y)返回 x 列表和 y 列表中相同的桶数 */
     SortC_ij=sortCount(Count_ij)
/* 函数 sortCount(x)返回降序排列的 x 列表 */
     end foreach
   end foreach
```

4.4 图稀疏化处理

这一步骤就是根据结点 v_i 所有的邻接边的匹配桶数 $SortC_{ij}$，使用一组 Map 任务选取前 d_i^e 条边，将其保留下来。其中 d_i 表示的是 v_i 结点的度，$e<1$ 是用户自定义的图稀疏化指数。这里用户可以通过改变 e 的大小来很方便地控制整个图的稀疏化程度。通过计算可以得出，e 越大，图稀疏程度越低，反之，图的稀疏程度越高。

具体过程如图 6 所示。

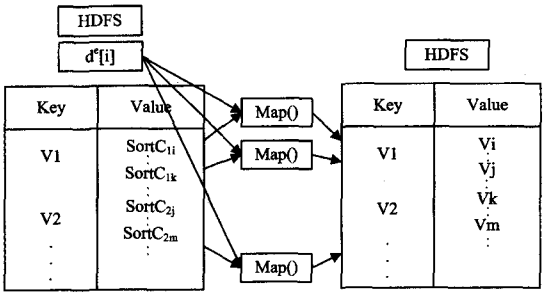

图 6 保留结点过程

这一步骤中，Map 任务是对上一个步骤的输出〈$key=v_i$，$value=list[SortC_{ij}]$〉进行计算，同时将 d_i^e 也作为输入，得到结果〈$key=v_i$，$value=list[top]$〉，其中 $list[top]$是保留 $list[SortC_{ij}]$列表的前 d_i^e 条边的列表。最后将结果输出到 HDFS 上。

此步骤的 Map 任务用形式化描述如下：

Map：

〈$key_1=v_i$，$value_1=list[SortC_{ij}]$〉

→〈$key_2=v_i$，$value_2=list[top]$〉

图的稀疏化处理步骤具体处理过程如下所示：

输入：上一步骤输出的键值对〈v_i，$list[SortC_{ij}]$〉，d_i^e中 e 为用户定义的图稀疏的指数，d_i 为边的度。

输出：〈v_i，$list[top]$〉，其中 value=list[top]表示需要保留下来的结点的邻居结点。

```
1. list[top]←ϕ/* 初始化结点需要保留的邻居结点集 */
2. foreach v_i in Graph do
     top=ToSave(list[SortC_ij],d_i^e)
/* 函数 ToSave(x,y)返回 x 列表中前 y 条边，并且根据边找到其所
含的结点，记录下来 */
   end foreach
```

通过以上步骤，图中各个结点都保留了至少一条边，这样确保了整个图还是一个连通的图。

5 实验结果及分析

本节通过实验验证了 MR-GSpar 算法的性能与效率。

5.1 实验配置

本文使用开源的 MapReduce 框架，将它部署到 Hadoop 集群上。本集群由 6 台机器组成，其中 1 台作为 master，5 台作为 slave，集群中每个节点的 CPU 频率均为 3.20GHz、Intel 双核，内存容量为 2GB。集群平台中采用的是 Hadoop1.0.3，操作系统为 ubuntu11.04，实验的代码均由 java 编写。实验用的数据源为 Twitter 社交网络的朋友关系图。

实验中采用算法加速比（Speedup）作为评价算法性能和效率的指标。加速比公式为：$S_{speedup}=T_i/T_1$，其中 T_i 为 i 个集群结点对原图稀疏化所需要的时间，T_1 为单个机器对原始图稀疏化需要的时间。

5.2 实验结果及分析

本文算法中涉及到稀疏比率 e，设置不同的系数比率得到的稀疏图不同，可以根据不同的需求设置不同的 e。不同数据量和不同类型的数据，其最佳 e 值也不相同。这里暂且设置为 $e=0.15$。

为更好地表明 MR-GSpar 算法的优势，实验中将利用不

（下转第 212 页）

误的类别信息被用于指导解码，造成系统误识率轻微上升。

系统3用元音稳定段排除了解码时部分不符合声学实质的语音候选段，避免了由此产生的解码错误，因此在提高运行时间的同时，识别系统的错误率有所降低。

最后将界标点提供的所有信息同时应用于随机段模型系统，结果如表6所列。

表6 所有信息引导的解码算法识别结果

系统类型	WER(%)	解码时间(min)
基线系统	13.67	24.0
系统4	14.01	20.9

系统4将声韵母边界信息、类别信息以及元音稳定段同时用于指导随机段模型的解码。与基线系统相比，系统4的系统误识率轻微上升，解码时间下降了12.9%。

结束语 根据声门界标点分析语音单元的边界信息、类别信息以及元音稳定段，并将上述语音学信息用于指导随机段模型的解码。实验结果表明，本文提出的方法能够有效地提高段模型大词汇量汉语连续语音识别系统的解码速度。接下来的研究工作将着重提高其它类型声学界标点的检测精度，并分析不同类型声学界标点相结合时蕴含的语音学信息，将其用于改进语音识别的解码。

参 考 文 献

[1] Kimball O, Ostendorf M, Bechwati I. Context Modeling with the Stochastic Segment model[J]. IEEE Trans. on Signal Processing, 1992, 40(6): 1584-1587

[2] 唐赟，刘文举，徐波. 基于后验概率解码段模型的汉语语音数字串识别[J]. 计算机学报，2006，29(4)：635-642

[3] Chao Hao, Yang Zhan-lei, Liu Wen-ju. Improved Tone Modeling by Exploiting Articulatory Features for Mandarin Speech Recognition[C]// Proceedings of ICASSP. 2012: 4741-4744

[4] Tang Yun, Liu Wen-ju, Zhang Hua. One-pass coarse-to-fine segmental speech decoding algorithm[C]// Proceedings of ICASSP. 2006: 441-444

[5] Zhang Hua, Liu Wen-ju, Xu Bo. Research on Adaptive Step Decoding in Segment-Based LVCSR[C] // Proceedings of IEEE NLP-KE'07. 2007: 463-467

[6] 彭守业，刘文举，张华. 基于相邻段的随机分段模型解码算法及其在LVCSR中的应用[C]// 2008年全国模式识别学术会议. 2008: 432-436

[7] 张晴晴，潘接林，颜永红. 基于发音特征的汉语普通话语音声学建模[J]. 声学学报，2010，35(2)：261-266

[8] Yang Zhan-lei, Liu Wen-ju. A Novel Path Extension Framework Using Steady Segment Detection for Mandarin Speech Recognition[C]// Proceedings of InterSpeech. 2010: 226-229

[9] Liu S A. Landmark Detection for Distinctive Feature-based Speech Recognition[J]. Journal of the Acoustical Society of America, 1996, 100(5): 3417-3430

[10] Park C. Consonant Landmark Detection for Speech Recognition [D]. Massachusetts, Cambriage: Massachusetts Institute of Technology, 2008

[11] 唐赟. 基于随机段模型的汉语语音识别算法研究[D]. 北京：中国科学院自动化研究，2006

(上接第193页)

同Hadoop集群规模运行算法。本算法利用Hadoop加载不同的Map和Reduce算法过程，对图结构数据进行处理，即完成图数据的稀疏化。本文提出的算法加速比性能测试结果如图7所示。

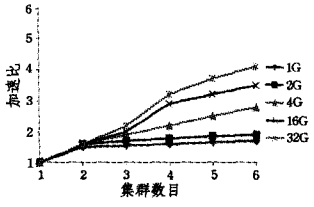

图7 算法性能测试结果

从图7中可以看出，尤其是面对大数据，利用Hadoop集群可以高效地减少时间的消耗，即加速比 $S_{speedup}$ 明显增大。根据Hadoop框架运行的特点，图数据量较大时，对图的稀疏化速率明显加快，有线性趋势；但是由于集群结点之间的数据通讯会有一定的数据开销，当图数据量较小时，图的稀疏化的优势会有所减弱，甚至需要的速率会略有下滑。同时加速比性能和集群的节点数成正比，即随着集群中节点数的增多，算法的加速比 $S_{speedup}$ 也逐渐渐增大。

上述结果表明，本文提出的基于MapReduce的图稀疏化算法更加适用于大规模图数据。即图数据量越大，算法的性能越好。其原因是算法设计中，增加了排序，合并了一些额外操作，使主节点和从节点之间的通讯代价大幅度减小，并且数据集规模越大，通讯量减少的比例越高。因此，当数据集规模越大时，算法的加速比性能越好。

结束语 本文深入研究了一种基于MapReduce的大图稀疏化算法。本文在介绍Minhash算法在图稀疏化处理中应用的基础上，在MapReduce计算框架上实现了Minhash算法的并行化改造，设计并实现了面向大规模图数据的分布式稀疏化算法，给出了具体的算法处理流程。最后，通过在多组不同大小数据集上的实验表明，本文提出的MR-GSpar算法适合运行于大规模并行平台下，提高了图稀疏化效率。

参 考 文 献

[1] Satuluri V, Parthasarathy S. Scalable graph clustering using stochastic flows: applications to community discovery[C]// ACM SIGKDD. 2009: 737-746

[2] Kulis B, Basu S, Dhillon I, et al. Semi-supervised Graph Clustering: A Kernel Approach[J]. Machine Learning, 2009, 74(1): 1-22

[3] Satuluri V, Parthasarathy S, Ruan Y. Local graph Sparsification for Scalable Clustering[C]// SIGMOD. 2011: 737-746

[4] 李建江，崔健，王聃，等. MapReduce并行编程模型研究综述[J]. 电子学报，2011，39(11)：2635-2642

[5] Lin J, Schatz M. Design patterns for efficient graph algorithms in mapreduce[C]// MLG. 2010: 78-85

[6] 尹丹，高宏，邹兆年，等. 一种新的高效图聚集算法[J]. 计算机研究与发展，2011，48(10)：1831-1841

[7] Lv Qin, Josephson W, Wang Zhe, et al. Multi-probe LSH: efficient indexing for high-dimensional similarity search[C]// Proc of the 33rd Int Conf on Very Large Data Bases(VLDB'07). Vienna Austria: VLDB Endowment, 2007: 950-961