基于 Hadoop 平台的 HITS 算法①

黄 婕

(空军航空维修技术学院 航空电子电气工程学院, 长沙 410014)

摘 要: 本文对 Hadoop 平台进行了分析研究后, 与 HITS 算法的设计理论和技术相结合, 对基于中文词汇网络的 HITS 算法进行了重新设计, 使其符合 Hadoop 平台的框架, 详细分析了 Map/Reduce 函数的设计方案. 用测试数据在不同集群上做实验, 实验结果证明, HITS 算法在分布式环境下能很好的运行, 集群的优越性明显.

关键词: Hadoop; HITS; 中文词; 算法

HITS Algorithm Based on the Hadoop

HUANG Jie

(Aviation Electronic and Electrical Engineering Institute, Air Force Aviation Repair Institute of Technology, Changsha 410014, China)

Abstract: In this paper HITS algorithm based on the Chinese vocabulary network is redesigned after analyzed the Hadoop platform, which is combined with the HITS's design theory and technology. This algorithm conforms to Hadoop framework.And Map/Reduce function's design scheme is analyzed in detail. With test data in a different cluster on the experiment, the results show that HITS algorithm can run well in a distributed environment and the cluster advantages are obvious.

Key words: Hadoop; HITS; Chinese vocabulary; algorithm

1 引言

随着网络信息时代的全面到来及计算机技术的急速发展, Web 这一超强大的数据资源库已逐渐形成. 网络上的 Web 信息包括页面本身的信息、访问 Web 页面的相关信息、超链接信息等. 怎样对丰富、广泛的各类资源信息进行管理、准确高效地获得有价值的信息是基于 Hadoop 平台 HITS 算法的一个核心且重要的任务.

云计算是网格技术、分布式技术的扩展、延伸[1], 有利于动态资源池、高可用性和虚拟化网络实验平台的规划[2], 在海量数据的存储、分布式程序开发和计算方面, 云计算这一先进技术有着很明显的优势. Google 的 GFS[3] 和 Map/Reduce 的算法模型与 BigTable[4]技术的出现, 受到了云计算爱好者的极大关注. Hadoop[5-7]这一分布式开源框架, 保证了用户方便、准确、有效的利用集群来计算存储数据, 且不必理会网络底层的分布式构架信息, 目前, Hadoop 的框架已被 Yahoo、Amazon 等众多大型网站重视.

2 Hadoop云计算平台的框架理论及技术

随着云计算的发展, Hadoop 这一网络平台也逐渐被更多人熟悉, Hadoop 源于 Apache Lucene 项目[8]中的 Apache Nutch 开源网络搜索引擎项目[9], 是一个能完成并行处理、开源的软件架构. Hadoop 通常在 Linux 平台上运行, 可以支持 java 编程语言、C 语言、C++ 等. 由于 Hadoop 的发展和进步, 它有着其他分布式系统不具备的特有的特征:

良好的可扩展性: Hadoop 框架在存储数据和计算方面都能扩展, 通过简单的配置, 就能实现集群的增加.

更经济的成本: 为了达到更加经济的成本, 数据和计算都是做的分布式并行处理, 且在 PC 服务器集群上来存储数据和计算操作.

高效率特征: 分布式系统能对文件系统进行并发, 有高效率的数据交互, 为了达到高效率, 实施不同点集群中的数据并行计算.

① 基金项目:湖南省教育厅科学研究项目基金(12C0920);空军航空维修技术学院项目基金(YC1208)
收稿时间:2013-07-25;收到修改稿时间:2013-11-04

高可靠性: 备份和恢复的分布式系统, 实行任务的分布监控, 对失效的任务进行重新部署, 保证了分布式数据的正确性、完整性和可靠性.

2.1 Hadoop 集群结构

Hadoop 的开源集群由一个 NameNode 和多个 DataNames 结点来构成的. 其中 NameNode 是 Master 结点, 由 JobTracker 来管理控制, 从而对文件系统的命名空间管理, 如打开、关闭目录文件, 对 DataNode 进行启动、调试、监测、关闭等任务, 并负责数据块在 DataNode 的映射, 同时客户端对文件系统的访问. 多个 DataNode 构成了集群 Slave, 他们在 NameNode 的集中统一控制下对数据块进行各种操作与计算, 一个 DataNode 就是一个结点, 每个 Slave 包含一个 DataNode 结点, 专门负责对 TaskTracker 监控, 对文件系统进行读写, 和数据在结点上的存储操作. 如下图 1, Hadoop 的集群的结构图.

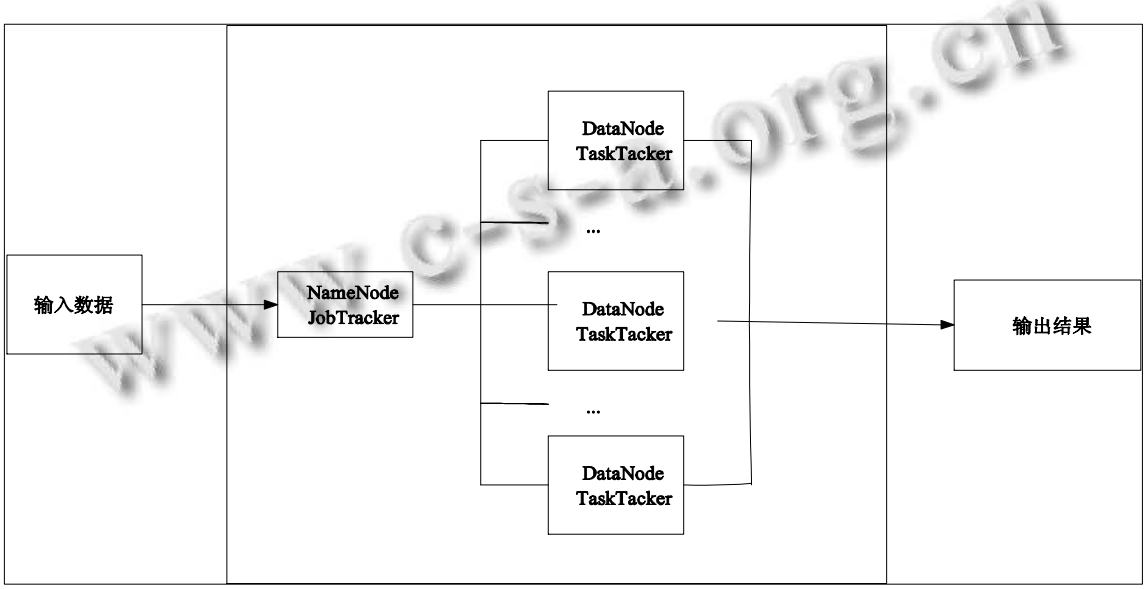

图 1 Hadoop 集群结点结构

满足 Hadoop 架构的计算和存储数据的两个功能, 其最重要的两大设计是:大规模进行数据计算和存储的 HDFS(Hadoop Distribute File System)和一个来源于 Google 论文[10]思想的 Map/Reduce 的计算模型.

2.2 Map/Reduce 的模型

Hadoop 的映射/规约(Map/Reduce)[11,12]计算模型是一种编程的规范, 可认为是对数据集进行分布式操作, 在 Map 阶段, 切分输入的数据, 每一个 Map 任务接收刚刚切分好的数据块, 同时 Map 任务得对 Key/Value 的数据对进行处理, 中间值即 Map 函数值, 这个中间值一般按照 Key 增量来顺序处理, 即可保证每个分区都按顺序输出文件. 若中间键值对重复比例大的时候, 就取本地数据集, 采用 Combiner 函数, 对相同的 Key 完成合并操作, 在网络上传输后, 给 Reduce 函数处理. 在 Reduce 阶段, 重点完成规约操作, 即对 Map 阶段的键值对重新组合, 再通过逻辑处理和交换, 形成较小的数据集. Reduce 的结果只有一个或者没有.

要保证在高风险的分布式系统下, 正确地运行, 则采用容错机制很必要, 在 Map 和 Reduce 阶段都有了容错的结构. 如运行中出现结点上的鼓掌, 服务器会对任务重新分配, 若机器上完成了部分的 Map 任务, 此时机器出现故障, 还需再次执行已经完成了的 Map 任务. 若完成了 Reduce 的任务, 则不需再次执行, 结果存储到全局文件系统中了.

3 HITS算法

HIFS 算法[13]是与 Web 有关的经典的数据挖掘算法, 也是一个动态查询相关的算法.

HITS 算法是对基于某个中心主题的搜索结果来分析排序的, 算法中引用 *Hub* 值和 *Authority* 值是为了利用页面的引用和被引的次数, 该次数可对网页进行重要性的测评.

Hub: 表示该页面的出度值, 即指向其他页面的数量.

Authority: 表示该页面的入度值即页面, 即页面被引次数.

一般入度值 Authority 较高的页面会被 Hub 值较高的页面所指向, 而出度值较高的页面也指向较好的入度值页面, 两个值相互作用, 相互促进, 这样有利于挖掘权威的页面和高质量的页面. HITS 的基本原理就是利用 *Authority* 值和 *Hub* 值来判断网页的重要性.

HITS 算法是从搜索查询的关键字或主题 Q 出发, 得到初始的查询结果, 然后选择结果集的 K 个页面, 称 K 为根集(Root Set), 记作 *R*. *K* 集合包含的是与查询相关的页面, 其中主要有被中心页面引用的链接和较权威链接, 因此对 Root Set 中的页面来扩展, 被扩展的页面叫基集(Base Set), 记作 *S*(*S*<=*m*, *m* 是基集页面扩展的上限).

用 Web 链接的有向图 *G*(*V*, *E*), *V* 是顶点集合, *E* 是有向边的集合, 若 $u,v \in V$, 且有向边 $(u,v) \in E$, 即可表示页面 *v* 有链接指向 *u*; *v* 的出度表示该页面指向另外页面的数量, 页面 *u* 的入度值是指向该页面的数量. 其中每个页面 *p* 都有个 *Authority* 的初始值 *a*(*p*), *Hub* 的初始值 *h*(*p*), 一般情况下, *a*(*p*)和 *h*(*p*)是一个非负常数, 若取值为 1.

$a(u)$ 是页面 *u* 的 *Authority* 值, 即:

$$a(u) = \sum_{(v,u)\in E} h(v) \tag{1}$$

$h(v)$ 是页面 *v* 的 *Hub* 值, 即:

$$h(v) = \sum_{(v,u)\in E} a(u) \tag{2}$$

对 $a(u)$ 和 $h(v)$ 进行规范化后得:

$$a(u) = a(u) / \sqrt{\sum_{\substack{(v,u)\\ q\in v}} [h(q)]^2} \tag{3}$$

$$h(v) = h(v) / \sqrt{\sum_{\substack{(v,u)\\ q\in v}} [a(q)]^2} \tag{4}$$

相对于原始的 $a(u)$ 和 $h(v)$ 的值规范后得公式(5)、(6), 公式(3)、(4)是对页面进行规范化了的, 是算法的改进.

$$a(u) = a(u) / \sqrt{\sum_{q\in v} [a(q)]^2} \tag{5}$$

$$h(v) = h(v) / \sqrt{\sum_{q\in v} [h(q)]^2} \tag{6}$$

从公式(5)、(6)的规范化的处理来看, 当涉及一个结点的 *Authority* 值和 *Hub* 值, 要获得整个网络结点的信息. 对于结点 *u* 的 *a*(*u*)值, 只于该结点的 *Hub* 值有关; 而 *v* 的 *h*(*v*)值, 只于该结点的 *Authority* 值相关. 在规范化后, 该计算相关结点的小网络群体的信息将替代所有结点的网络信息, 这样有利于提高算法的运行效率.

4 基于中文词的HITS算法

中文词网络中的词语的前后位置和 Web 结构图中的页面指向与被指向关系对应, 每个词语在网络中都有出度和入度. 把 HITS 算法移植到中文网络词中搜索研究, 每个中文词都有自己的 *Authority* 值和 *Hub* 值, 该定义如下:

Authority 值: 表示该词的入度值, 即该词语的入度列表中的词语的个数.

Hub 值: 表示该词的出度值, 即该词的出度列表中的词语的个数.

一词语的入度值和出度值的定义, 在词网络中, 因语言的特征, 语义相关性使得关键词与前后语义环境息息相关, 与网络连接结构有很大的相似性, Web 页面之间都存在一定的链接关系. 若一个词语在其出度列表中有好的值, 则会有好的 *Hub* 值. 若词语的 *Authority* 值较高, 则该词语的入度列表的词语就有好的 *Hub* 值, 利用这种词语间的相互加强关系, 来获取有利的中文词语的资源. 在以下的中文语义词汇网络图中, 词语的出度 *Hub* 值和入度 *Authority* 值的关系如下图 2 表示.

5 基于Hadoop的HITS算法

该任务的主要目的是建立基于中文词汇网络的 HITS 算法, 并根据设计好的方案进行试验.

5.1 设计方案

基于网络中文词的 THIS 算法, 需要 *Authority* 和 *Hub* 值. 设计利用词语出现的先后顺序, 把每个词语看成词语网络中的结点, 将相邻结点连线. 传统的 HITS 算法是在技术词语的 *Authority* 值和 *Hub* 值是, 用矩阵来存储结点信息, 其中矩阵的行、列值是结点的数目相同. 而基于 Hadoop 的中文词的 HITS 算法, 是在计算了 *Authority* 值和 *Hub* 值规范化后, 若在 Hadoop 平台上, 依据公式(5)、(6)来处理, 无法依据每个结点的信息来获取相关结点的出度信息. 因此必须采用 Map/Reduce 来完成词语的 *Authority* 值和 *Hub* 值的规范化. 因此, 一次 HITS 的完整的迭代就需要两次

Map/Reduce 的操作. 即第一次的 Map/ Reduce 的操作是计算相关词语的 *Authority* 值和 *Hub* 值, 第二次的 Map/Reduce 的操作, 是对这两个值的规范化.

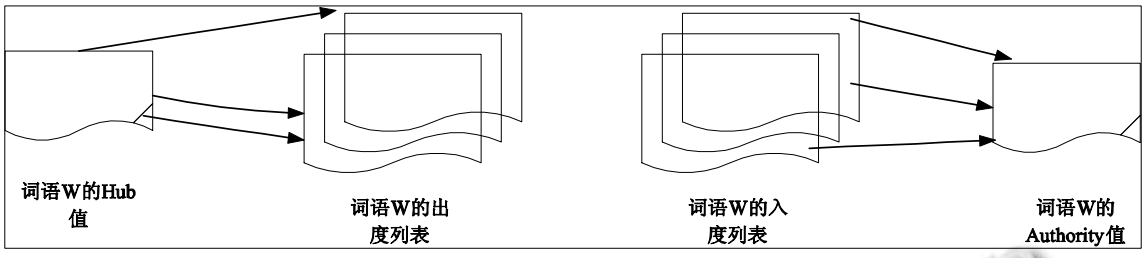

图 2 词语的 Authority 值和 Hub 值关系

以下的实验, 利用改进的公式(3)和(4)对 *Authority* 值和 *Hub* 值进行规范化处理. 在 Reduce 阶段, 当映射输出时有该结点的出度、入度值时, 就能对 *Authority* 值和 *Hub* 值进行规范化. 先计算值, 然后再规范化.

在 Hadoop 平台上实现基于中文的 HITS 算法的计算过程, 用一张网络链路图表示中文词的结构图. 图中的 *n*1, *n*2, , , *n*7 是有先后顺序的, 包含的是有向边. 如下图 3 所示.

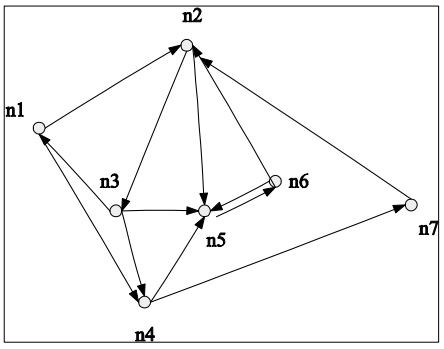

图 3 中文词网络

(结点表示词语, 有向边表示词语间的先后顺序)

以图 3 为例, 构造词语的出度入度列表, 如下表 1 所示.

表 1 词语的出度、入度列表, 结点表示词语

结点表示的词语	该词的出度	该词的入度
n1	(n2, n4)	[n3]
n2	(n3, n5)	[n1, n6, n7]
n3	(n1, n4, n5)	[n2]
n4	(n5, n7)	[n1, n3]
n5	(n6)	[n2, n3, n4, n6]
n6	(n2, n5)	[n5]
n7	(n2)	[n4]

Hadoop 平台的 HITS 基于中文词的算法, 应当尽量把词汇信息的表达方式设计成符合的设计方案, 是算法的重点, 我们把算法的映射规约阶段分开探讨, 图 4 是 Map 的流程.

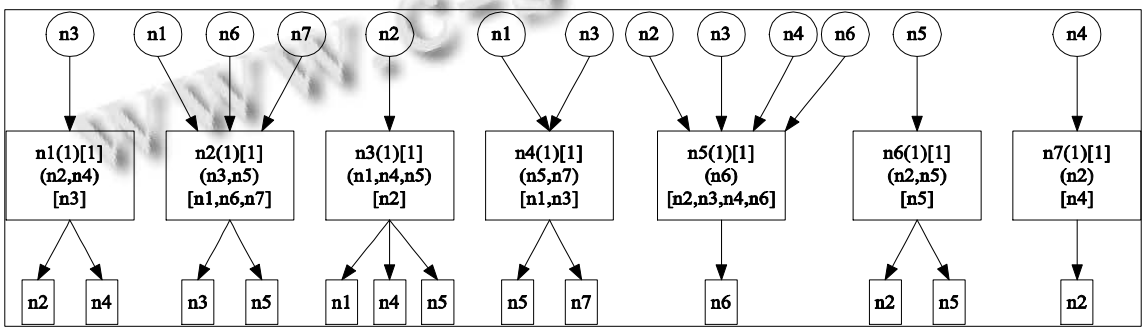

图 4 HITS 的映射 Map 流程

在图 4 中, 中间矩形表示结点的信息, 包括节点名称、小括号中的 *Authority* 值, 中括号的 *Hub* 值; 第二行表示结点的出度值, 第三行表示该结点的入度值. Map 阶段数据的分割单元是词汇的信息, 作为 Map 的输入. 在 HITS 算法中第一次求该结点的出度值 *Hub*、入度值 *Authority*, 必须要对出度值 *Hub* 和入度值

Authority 初始化. 在图中, 初始化为 1.

在映射 Map 阶段, 输入的数据是在 *NameNode* 结点上做并行运算操作, 把数据分割单元交给 *DataNode* 结点, 基于中文的 HITS 算法, 在 Map 阶段是为了求得词语 K 的出度列表的 N 的 Hub 中心值. 同时为保证在 Reduce 阶段的继续, 得存储运行时的上下文信息. 其中包括, 词语的出度入度值, Map 函数后的 Key/Value 中间值等.

Reduce 阶段, 利用 Map 函数的值作为输入的数据, 通过迭代计算, 求得每个词语的 *Authority* 值和 *Hub* 值, 而后对值来规范化. 图 5、图 6 是求得 *Hub* 值和 *Authority* 值时, 省略了规范化. 图 5 表示每个结点代表的词语的 Authority 值, 在 Reduce 阶段的具体过程. 其中, 每个结点具有新的 *Authority* 值, 第一个小括号是结点的入度信息, 第二个小括号是结点出度值. 计算 *Authority* 值是将结点的出度列表的结点按名称手机, 并把名称作为 *Key* 值, 其他信息是 *Value* 值. 图中的 *Authority* 值没有规范化.

图 5 表示的是结点表示的词汇的 *Authority* 值的 Reduce 阶段的操作.

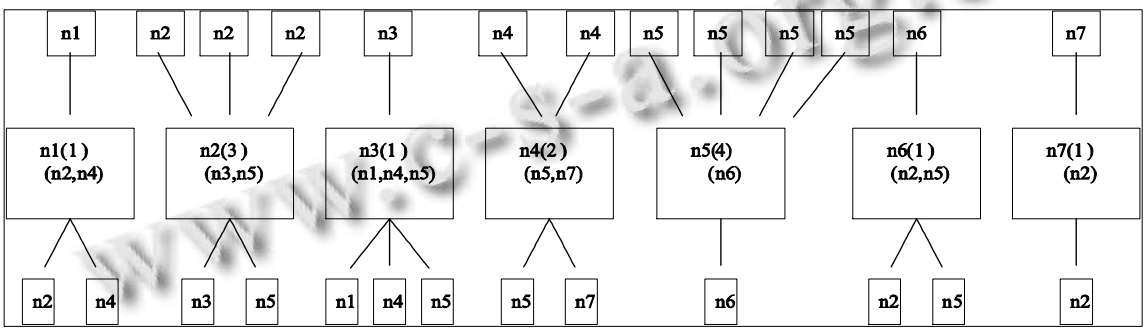

图 5　Reduce 阶段求词语的 *Authority* 值

图 6 表示 Reduce 期间求解 *Hub* 出度值的流程. 结点名称的第一个方括号里的值是表示该结点的新的 *Hub* 出度值. 该结点的入度信息则保存在第二个方括号里了. 计算 *Hub* 值是将结点的入读信息按名称收集, 作为 *Key* 值, 其他为 *Value* 值, 计算出来的 *Hub* 值没有规范化的.

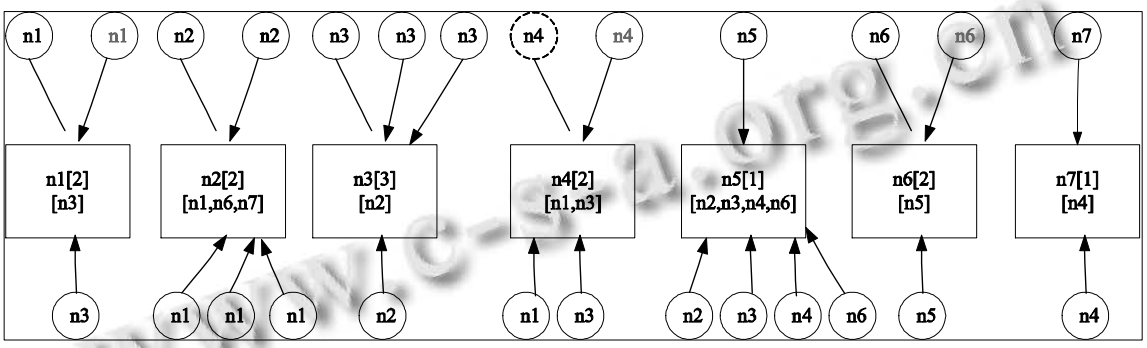

图 6　Reduce 阶段求词语的 *Hub* 值

一般情况下, 要取得稳定的 *Hub* 值和 *Authority* 值, 得从公式(1)和(4)反复进行迭代运算. 需要有效地判定算法的终止, 必须设计迭代方案, 因此, 实验中要设定一个阈值. 设阈值 threshold=10^{-4}, 迭代次数为 *k*, 结点 *i* 的 *Authority* 值记作 *Ai*(*k*)和 *Hub* 值记作 *Hi*(*k*),设计出本次迭代是 *k*+1 次, 如果 $|H_i(k+1)-H_i(k)|<threshold$ 且 $|A_i(k+1)-A_i(k)|<threshold$, 则随着迭代次数的增加, 各结点的 *Authority* 值和 *Hub* 值变化很小, 记作收敛标准, 程序终止, 迭代终止.

5.2 Map 函数的设计

Hadoop 平台的 Map/Reduce 框架, 将输入接收来的任意的文件和数据进行计算后, 以自定义的文件格式输出. 基于分布式自身的特征, 利用 Hadoop 的 Map/Reduce 框架, 将海量数据的文件分配到多个机器上, 这样机制保障了无限规模的集群运算.

基于中文词汇的 HITS 的 Map 操作, 要对数据格

式来做改变, 将 Map 数据划分单元, 如 mark2 所示.

word:
value-of-Authority<>value-of-Hub<>out-list<>in-list; (mark2)

word 表示中文词汇; value-ofAuthority 是 *Authority* 的值, value-of-Hub 是 *Hub* 值, out-list 是出度, in-list 是入度; <>表示数据分割符. 在计算 *Authority* 值和 *Hub* 值之前都得初始化, 值为 1. 表 2 是 Map 操作前后的键值对的输入输出值.

表 2 基于中文网络的 HITS 算法的 Map 操作前后的输入键值对和输出键值对

	Key/Value 对的输入	Key/Value 对的输出
Map Function	Key: <wordN,HubN,AuthN> Value: <out-listN ,in-listN>	Key:<wordM> Value:<wordN,HubN> Key:<wordH> Value:<wordN,AuthN> Output: wordN->out-listN wordN->in-listN

其中设词语的信息作为 Map 函数的输入, 表 2 中, 每个词语 N 以及该词汇的 *Authority* 值和 *Hub* 值都作为 *Key*, 该词的出度和入度作为 *Value*. Map 操作后, 把词汇 N 多输出列表中的 M 作为 *Key* 关键值, N 和 N 的 *Hub* 值作为 *Value* 值.

下列是 HITS 的 Map 伪代码.

```
(1) input<word N,Hub N,Auth N ><out-listN, in-listN >
(2) Nn_out= out-listN.length
(3) for (wordM = out-listN ;M<= (Nn_out-1);)
(4) output wordM-><wordN,HubN>
(5) output wordN=out-listN
(6) Nn_in= in-listN.length
(7) for (wordH= in-listN;H <= (Nn_out-1);)
(8) output wordH-><wordN,AuthN>
(9) output wordN->out-listN
```

5.3 Reduce 函数的设计

在执行 Reduce 函数时, 要收集相同词语的 Key, 表 3 中的 wordK 很多时候, 是出现在词语的出度列表, 又出现在入度列表中, 很少, 只出现在其中一个列表中. 当只出现在出度列表中, *Hub* 值为空 NULL; 当只出现在入度列表中, *Authority* 入度值为空 NULL.

表 3 基于中文词网络的 HITS 算法的 Reduce 操作前后的输入输出键值对

	Key/Value 对的输入	Key/Value 对的输出
Reduce Function	Key:<wordK> Value:<wordNi,HubNi> Key:<wordK> Value:<wordNj,AuthNj>	Key: :<wordK,AuthK,HubK> Value: <in-listK,out-listK>

在表中 Reduce 的输入是 Map 函数的输出数据格式, Reduce 完成一次后, 对每个词语的 Authority 值和 Hub 值都计算了, 还对两个值规范化了. 伪代码的每个词语 K 的 AuthKzhi, 出度列表中包含有 K 的完整 Ni 集合, K 的新的 *AuthK* 值是将 Ni 的 HubNi 所有值来求和得到. 依次计算每个 K 的 *HubK* 值, 求得入度列表中包含 K 的所有 Nj, 对 Nj 的 *AuthNj* 值累加, 得到 K 的新的 *HubK* 值. 然后利用公式(3)(4)规范化. 下面是基于中文词的 HITS 算法的 Reduce 伪码

```
(1) input word K-><wordNi,HubNi>   <out-listK>
(2) AuthK=0
  (3) for wordK –>out-listNi
(4) AuthK+=HubNi
(5) AuthK = AuthK / sqrt(sum_(Ni,k) (HubNi)^2)
(6) input wordK-><wordNi,AuthNi>
(7) HubH=0
(8) for wordK->in-listNj
(9) HubK+=AuthNj
(10) HubK = HubK / sqrt(sum_(k,Nj) (AuthNj)^2)
(11) Output <wordK, ,HubK AuthK > <out-listK, in-listK >
```

5.4 算法实验结果

实验是基于中文网络词汇的数据挖掘的基础上的, 首先对输入的原始的中文数据进行网络建模和, 再对数据进行分布式的 HITS 算法的计算. 实验采用多组数据来对比.

第一次实验:

数据为 460M, 运行 1 次、5 次和 15 迭代完后, 任意选取 15 个词汇, 求得在 HITS 算法中 *Hub* 值和 *Authority* 值如图 7, 图 8 X 轴表示中文词汇数目, Y 轴表示算法中的 *Hub* 值或者 *Authority* 值.

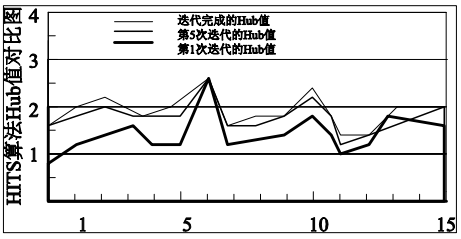

图 7 HITS 算法的 Hub 值的不同迭代次数的对比

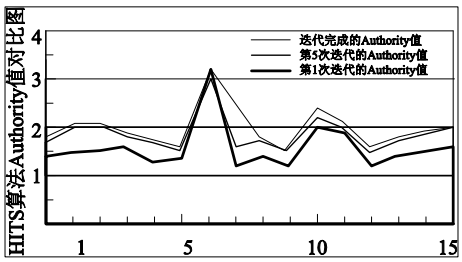

图 8 HITS 算法的 Authority 值的不同迭代次数的对比

上述两幅迭代次数对比图中，是利用不同词汇在同以一次的迭代中的 *Hub* 值和 *Authority* 值用折线相连，从图示来看，分布式的算法和传统的单机的 HITS 算法类似，因此，*Hub* 值和 *Authority* 值还要加强.

第二次实验:

是将数据复制 10 遍，变为 4600M，将数据拷贝两份，一份在集群上，在迭代完成计算后，如表 4 中所示;另一份拷贝数据存储在 Hadoop 上，在分布式的基础上完成迭代. 如表 5 记录运行完成时间.

表 4 HITS 算法在集群上的 3 次迭代(数据位 4600M)

集群数目	迭代完成计算的时间(s)	节约的时间(s)
1	18716	0
2	15708	3109
3	14691	6016
4	11953	8664

表 5 HITS 算法在集群上迭代计算的完成时间

集群数目	迭代 3 次的时间(s)	节约的时间(s)
1	3844	0
2	4283	-429
3	3329	506
4	2821	1014

HITS 算法运算在 DateNode 结点上，以 3 次算法迭代的完成时间作为参考点. 当集群数量是 1 台是，完成时间是 3943s，为 2 时，是 4381s. 由此可见，集群的数量在增加，计算能力加强了，但是执行的时间延长，原因是增加了网络间的通信流量.

实验证明，HITS 算法的迭代次数增加，数据的计算量同时增加，集群的优势就更明显.

6 结语

基于中文词汇的 Hadoop 平台的 HITS 算法，有下列特征:

Hadoop 平台的 HITS 算法有一定可行性. 数据规模比较小的时候，集群数量的增长，会使得运行效率降低，但随着数据规模的增长，算法的效率也有一定的提高，其原因是存在集群的数据通信时间.

数据在集群上的有效运行，使得集群能发挥更大的优势. 利用算法对 Hub 值和 Authority 值做规范化处理时，不需要网络信息就能在无需其他信息，而仅有本结点的信息就能完成计算，该算法是有利于分布式运行的，因此 HITS 算法在集群上运行很有优势.

参考文献

1 Vaquero LM, Rodero ML, Caceres J. et al. A break in the clouds: Toward a cloud definition. ACM SIGCOMM Computer Communication Review, 2011,39(1): 50–55.

2 陈康,郑纬民.云计算:系统实例与研究现状.软件学报,2009,20(5):1337−1348.

3 Ghemawat S, Gobioff H, Leung ST. The Google file system. Proc. of the 19th Annual ACM symposium on Principles of Operating System. New York. 2003. 29–43.

4 Chang F, Dean J, Ghemawat S et al. BigTable: A distributed storage system for structured data. Proc. of the 7th USENIX Sysposium on Operating System Design and Implementation. 2006. 205–218.

5 White.曾大聃,周傲英译.Hadoop 权威指南.北京:清华大学出版社,2010.

6 Venner J. Hadoop scalable distributed applications in the cloud. United States of America. Apress. Dec.2011.

7 Hadoop. Documentation and open source release. http://hadoop.apache.org/core.

8 Apache Lucene. Welcome to Apache Lucene. http://lucene.apache.org/.

9 Apache Nutch. Welcome to Nutch. http://nutch.apache.org/.

10 The Hadoop Distributed File System. Architecture and Design. http://hadoop.apache.org/core/docs/current/hdfs_deisgn.html

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

12 Lammel R. Google's MapReduce programming model-revisited. Science of Computer programming, 2012, 70(1): 1–30.

13 Kleinberg JM. Authoritative source in a hyperlinked Environment. Journal of ACM, 1999, 46(5): 604–632.