基于 HAMA 的半监督哈希方法①

刘 扬, 朱 明

(中国科学技术大学 自动化系, 合肥 230027)

摘 要: 在海量数据检索应用中, 基于哈希算法的最近邻搜索算法有着很高的计算和内存效率. 而半监督哈希算法, 结合了无监督哈希算法的正规化信息以及监督算法跨越语义鸿沟的优点, 从而取得了良好的结果. 但其线下的哈希函数训练过程则非常之缓慢, 要对全部数据集进行复杂的训练过程. HAMA 是在 Hadoop 平台基础上, 按照分布式计算 BSP 模型构建的并行计算框架. 本文尝试在 HAMA 框架基础上, 将半监督哈希算法的训练过程中的调整相关矩阵计算过程分解为无监督的相关矩阵部分与监督性的调整部分, 分别进行并行计算处理. 这使得使得其可以水平扩展在较大规模的商业计算集群上, 使得其可以应用于实际应用. 实验表明, 这种分布式算法, 有效提高算法的性能, 并且可以进一步应用在大规模的计算集群上.

关键词: 无监督哈希算法; BSP 模型; 分布式计算; Hadoop 平台; HAMA 框架; 矩阵计算

HAMA-Based Semi-Supervised Hashing Algorithm

LIU Yang, ZHU Ming

(Department of Automation, University of Scince and Technology of China, Hefei 230027, China)

Abstract: In the massive data retrieval applications, hashing-based approximate nearest(ANN) search has become popular due to its computational and memory efficiency for online search. Semi-supervised hashing (SSH) framework that minimizes empirical error over the labeled set and an information theoretic regularizer over both labeled and unlabeled sets. But the training of hashing function of this framework is so slow due to the large-scale complex training process. HAMA is a Hadoop top-level parallel framework based on Bulk Synchronous Parallel mode (BSP). In this paper, we analyze calculation of adjusted covariance matrix in the training process of SSH, split it into two parts: unsupervised data variance part and supervised pairwise labeled data part, and explore its parallelization. And experiments show the performance and scalability over general commercial hardware and network environment.

Key words: semi-supervised hashing; bulk synchronous parallel mode; hadoop; HAMA framework; matrix computation

1 介绍

近些年来, 互联网数据增长非常迅速, 尤其是在文档、图片、视频等方面. 在海量数据中, 获取相关的信息的需求越来越严峻. 目前基于文档的搜索引擎已经得到普遍的应用, 如 Google, 百度等. 基于内容的图片搜索, 成为下一个信息检索的需求点. 在过去的十年中已经进行了广泛的研究. 基于内容的图片搜索, 一般直接以可视的请求 q, 尝试从一个给定的, 样本个数为 n 的数据集 X 中检索, 以特征空间中, 特定距离标准下的最近邻, 作为相似结果返回[1]. 最近邻算法, 一般复杂度为 O(n), 因为需要比较数据集 X 中的所有样本到请求 q 之间的距离. 而实际中许多应用中检索时间对用户体验有着直接性的影响, 因而非常关键. 实际上用户一般会同时检查多个返回结果, 而且检索算法的本身的准确度有限, 因此返回近似最近邻在实际应用就可以了. 对于近似最近邻(ANN), 使用基于树的算法[2], 如 KD 树、球树、度量树、优点树等等, 则算法复杂度可以降为 O(log |X|). 但是一般基于树的算

① 基金项目:中国科学院重点部署项目课题(KGZD-EW-103-5(5))
收稿时间:2014-03-23;收到修改稿时间:2014-05-04

© 中国科学院软件研究所 http://www.c-s-a.org.cn

法，存在内存的限制，而且对于高维数据检索，因为维数灾难问题性能会有严重的下降.

在这种情况下，引入了基于哈希的近似最近邻算法. 关于哈希算法的流程见图 1. 这种算法在线检索时候，一般只需要较小的存储空间以及常量的检索时间. 而基于哈希的近似最近邻算法分为两类，监督方法与无监督方法. LSH，是最为广泛使用的无监督哈希算法. 但是，近似最近邻只能够返回近似结果，同时由于高层视觉元素的语义描述与底层特征描述的巨大差距，无监督算法，不能够保证较好的返回结果. 如果存在数据集中样本相似或者不相似的标记信息，那么哈希算法则生成一个满足该标记信息的哈希函数. 这就是监督方法的来源. 实际中应用使用者的反馈信息可以得到这种标记信息. 相比于无监督方法，监督方法需要一个缓慢的训练过程.

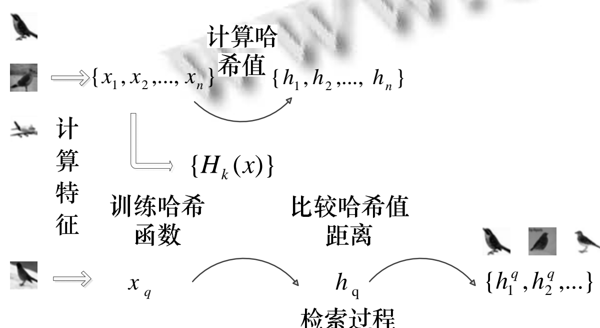

图 1 基于哈希的最近邻算法过程

半监督方法[3]对的目标函数包含两个部分，监督性的经验数据拟合和无监督的信息理论正规化. 这样通过经验数据的标记信息跨越从特征到信息的语义鸿沟，避免过拟合，保持结果的鲁棒性. 关于半监督哈希算法的理论分析过程见后面的讨论. 半监督哈希算法主要分为线下哈希函数计算与线上检索阶段. 根据前面的讨论，半监督哈希算法的线上检索非常有效，而且占用内存较少. 但是其线下哈希函数计算部分，则非常缓慢. 其中特征计算部分，是完全并行的. 根据特征训练哈希函数的过程，因为涉及到大规模矩阵计算，一般非常缓慢. 在实际应用中，可以考虑通过分布式方式，对其进行加速.

并行程序设计，一般而言需要考虑很多因素，如容错，负载均衡，同步等细节问题. 很多分布式计算的方法如 MPI 或者 CUDA 等，或者受限于单机的计算能力，或者需要专门的高性能计算集群. 谷歌提出的 mapreduce 计算模型，简化了并行程序设计的过程，提供了分布式程序的基本框架. 该框架在提供水平扩展计算能力的同时提高了系统的容错性，使得大规模分布式计算得以在商用的不可靠硬件上进行，而不再是需要专门建立高性能计算集群. Hadoop[4]是 mapreduce 的开源实现.

设计论证顺序算法，采用传统的冯诺依曼模型. 而在设计并行算法时候，一般有 PARAM 模型和 BSP 模型[5]. 而在[6]中，讨论了在 mapreduce 模型基础上建立 BSP 计算模型，以及在 BSP 模型上建立 mapreduce 模型的问题. HAMA[7]是 Hadoop 项目的子项目. 它在 Hadoop 的平台基础上，建立基于 BSP 模型以及 graph 模型的计算框架. Hadoop 本身的计算模型为 mapreduce，关于 BSP 模型，以及如何在 Hadoop 的计算框架下实现 BSP 计算模型，后面有具体的讨论.

文章的剩余部分结构如下：第 2 节，具体介绍半监督哈希算法. 第 3 节，介绍 BSP 模型以及 HAMA 项目. 第 4 节，讨论在 HAMA 中实现半监督哈希算法的关键过程. 第 5 节，实验. 第 6 节，总结.

2 半监督哈希算法

根据[3]，给定数据集 $X=\{\mathrm{x_i}\}, \mathrm{i}=1,2,...,\mathrm{n}$，其中 $x_i \in \Re^D$. 对于给定的 q，在 X 中，进行其近似最近邻搜索. 标记信息则使用 S 表示.

$$S_{ij}=\begin{cases}1 & (x_i,x_j)\text{为同一类型}\\ -1 & (x_i,x_j)\text{为不同类型}\\ 0 & (x_i,x_j)\text{情况未知}\end{cases},$$

可以看到 $S^T=S$,S 是对称阵.

根据半监督哈希算法，目标函数

$$J(W)=\frac{1}{2}tr\{W^T[X_lSX_l^T+\eta XX^T]W\}$$
$$=\frac{1}{2}tr\{W^TMW\}$$

这里 M 为调整相关矩阵.

$$M=X_lSX_l^T+\eta XX^T=M_l+\eta M_C$$

其中 $M_C=XX^T$ 表示无监督数据相关性部分，通过 $M_l=X_lSX_l^T$ 成对标记信息进行调整.

求解 $\arg\max_W J(W)$ 则得到哈希函数. 其中第 k 个哈希函数如下:

$$h_k(x)=\mathrm{sgn}(w_k^Tx+b_k)$$

从 M 求解到 W 的过程，有正交映射方法，非正交映射方法，序列映射方法等. 以上方法都是对于 M 矩

© 中国科学院软件研究所 http://www.c-s-a.org.cn

阵, 或者求解器特征矩阵, 或者增加非正交因子, 然后对其进行 LU 分解等, 或者根据 Mc 求解得到 W, 然后逐渐通过 S 来对其进行训练, 使得其满足更多的 S 标记矩阵情况. 这部分的操作都是集中在 M 矩阵上进行.

调整相关矩阵 $M \in \Re^{D \times D}$ 的计算过程, 主要是矩阵乘法与加法. 随着数据集合, 快速增长. 而且占用的内存, 也在线性增长. 因此下面的分布式计算主要集中在矩阵的乘法阶段. XX^T 部分, $X \in \Re^{D \times n}$, 算法复杂度为 $O(D^2 n)$. $X_l S X_l^T$, 只需要计算相应的 $S_{ij} \neq 0$ 的位置即可. 复杂度为 $O(D^2 l)$, 其中 l 为 $S_{ij} \neq 0$ 的元素个数. 具体复杂度的比较见表 1. 可以看到上述的 M 的计算过程, 还是有很多其特殊性的. 可以针对其进行特别的优化, 具体见后面的分析.

表 1 调整相关矩阵 M 相关运算复杂度

XX^T	$X_l S X_l^T$	$M = M_l + \eta M_C$	$eig(M)$
$O(D^2 n)$	$O(D^2 l)$	$O(D^2)$	$O(D^2)$

对于 M 矩阵之后的计算过程, 因为 $M \in \Re^{D \times D}$ 与 n 或者 l 无关, 因此算法复杂度不因数据集合的增加而增加. M 的特征根运算, 其为对称实矩阵, 因此可以直接采用 Jacobi 方法[8], 求解其特征根与特征向量. Jacobi 方法, 每进行一次 Givens 旋转变换, 影响的元素个数为 2D 个, 一般进行 D 次即可, 其算法复杂度为 $O(D^2)$. 对于 M 的其他操作, 如 LU 分解等运算, 也可以进行类似分析. 对 M 的操作, 都与数据集的增长 $n = |X|$ 无关. 实际中, 尽管维度 D 可能较高, 依然采用单机进行计算即可.

3 BSP模型与HAMA框架

BSP 模型[5], 即整体同步并行计算模型. 一个 BSP 计算机, 拥有 p 个处理器, 每个处理器有自己的存储空间, 处理器之间通过点对点的方式进行通信. BSP 算法由一系列的超级步组成. 在一个超级步开始时, 处理器接收输入, 之后 p 个处理器异步地进行计算, 在超级步结束时候, 处理器将输出进行通信. 在一个超级步结束的时候, 使用同步屏障来保证 p 个处理器之间完成信息的同步.

在 HAMA[7]中, BSPMaster 作为主节点, 负责任务的接受与分配. 每个计算节点上则运行着 GroomServer 以及 ZooKeeper[9]. BSPMaster 在收到任务请求后, 请求一个 GroomServer 划分任务, 然后将任务分配到若干个 GroomServer 上. GroomServer 根据任务的情况开启 BSPPeer, 作为 BSP 运算模型中的处理器单元. BSPPeer 进行具体的计算任务. BSPPeer 之间通过 ZooKeeper 相互通信, 并且进行屏障同步, 保证各个节点之间协同计算. Zookeeper 提供了一个分布式协作系统, 用于实现分布式应用, 这里是作为一个分布式屏障来使用.

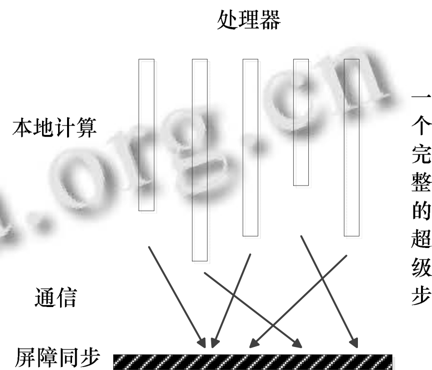

图 1 BSP 计算模型

BSPPeer 则通过 send、receive 方式进行沟通. 一个 BSP 任务都会继承自 BSP<K1, V1, K2, V2, M>类. BSP 任务的输入输出都是按照键值对<Key, Value>方式. 这种键值对的方式, 主要是因为 HAMA 是建立在 mapreduce 框架上的, 而在 mapreduce 框架中输入都是典型的键值对结构. 而在 HAMA 中, 一个完整的程序可能由多个超级步组成, 因此 BSP 超级步的输入输出都是键值对结构. 其中<K1, V1>是输入数据的键值对类型, <K2,V2>为输出键值对类型, M 为每个超级步结束前各个 BSPPeer 之间进行发送消息的消息数据类型. 关于如何利用键值对这种结构进行输入输出, 以及一个典型的 BSP 任务, 具体见后面的讨论. HAMA 框架中已经有实现基本的矩阵乘法的示例[13].

4 HAMA框架中的矩阵运算

从第 2 节可以看到, 半监督哈希算法的运行效率的关键在于矩阵的乘法运算的效率. 而矩阵的其他运算, 如特征值以及 LU 分解等相对重要性较低. 下面具体讨论如何在 BSP 模型下计算 M 矩阵.

4.1 半监督哈希算法中矩阵计算分析

早期在 HAMA 中, 矩阵以数据库形式存储在 HBase 中. HBase 是建立在 HDFS 之上的分布式数据库系统. 矩阵存储在 HBase 中按照行向量进行存储的.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

这种方式相对比较复杂, 因为需要专门设计数据库的表, 同时也限制了矩阵的表示方式. 现在在 HAMA 框架中, 矩阵以 SequenceFile 文件形式存储在 HDFS 中. 具体矩阵的存储形式, 并不是唯一或者确定的, 可以根据应用程序的需要设置和转换.

在密集矩阵相乘情景下, 一般通过行号: 行向量方式存储. 该数据的格式一般为键值对 IntWritable: VectorWritable 格式, 存储在 SequenceFile 中. 但是这样设计, 有一个细节问题. 对于

$$\begin{aligned} C &= AB \\ &= [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \dots \quad \mathbf{a}_n]^T [\mathbf{b}_1 \quad \mathbf{b}_2 \quad \dots \quad \mathbf{b}_m] \end{aligned}$$

可以看到自然的存储方式是按照行向量存储 A, 而利用列向量存储 B. 这样在矩阵的乘法计算过程中, 直接计算 $c_{ij} = \mathbf{a}_i \mathbf{b}_j$. 因此, 注意 B 的存储方式, 也就是存储 B 的列向量. 这与存储 B 的转置的行向量, 本质是一样的. 这里记为 $C = AB = A \otimes B^T$,其中 $A \otimes B^T$ 即表示两矩阵的对应行向量相乘的运算. 这样矩阵的乘法运算, 必须首先进行 B 矩阵的转置运算才可以. 特别的, 对于矩阵与自己的转置相乘, 这一操作, 对于 $C = AA^T = A \otimes A$, 就可以直接跳过矩阵的转置预处理, 直接保存一个 A 就可以. 在半监督哈希算法, 相应的 $M_C = X \otimes X$. 因此这种存储方式, 对于计算矩阵与其自身转置相乘的情况下, 反而是简单的.

而对于监督部分 $M_l = X_l S X_l^T$, 这里的 S 一般为稀疏矩阵, $l << n^2$,

$$\begin{aligned} M_l &= X_l S X_l^T \\ &= [x_1 \quad x_2 \quad \dots \quad x_m][S_{ij}][x_1 \quad x_2 \quad \dots \quad x_m]^T \\ &= 2[\sum_{(x_i, x_j \text{同一类型})} x_i x_j - \sum_{(x_i, x_j \text{不同类型})} x_i x_j] \end{aligned}$$

这里 S 为稀疏矩阵, 为了方便计算 $\boldsymbol{M_l}$, 对于 S 按行存储每个行向量. 每个行向量中仅存储其对应元素不为零的位置以及其符号. 对于 S 中每个不为零的元素 S_{ij}, 由特征向量组成的矩阵 $\boldsymbol{X_l}$ 中将相应的列 x_i 与 x_j 相乘, 然后就可以得到结果. 因此这里对 $\boldsymbol{X_l}$ 矩阵按照列存储.

4.2 BSP 算法过程

根据上面的理论分析, 对于 $M_C = X \otimes X$ 计算, BSP 算法如下:

BSP 过程输入: 行号 i, 行向量 x_i

算法:

For j = 0: i

读取 X 对应 SequenceFile 第 j 行数据, 即 x_j.

计算 $x_i x_j$, 得到 M_{Cij}

End

得到 $[M_{C0}.M_{C1} \quad \dots \quad M_{Ci}]$ 部分.

BSP 过程输出行号 i, $[M_{C0}.M_{C1} \quad \dots \quad M_{Ci}]$ 行向量到主节点.

主节点得到 M_C 的下三角部分. M_C 对称, 即得到 M_C 矩阵.

算法结束.

对于 $M_l = X_l S X_l^T$ 计算如下:

BSP 过程输入: 列号 i, 列向量 x_i

算法:

初始化 M_l 矩阵为 D*D 的全零矩阵

读取 S 矩阵的第 i 行 S_i

For j = 0: n

读取 S_i 第 j 列 S_{ij}, 如果不存在(也就是为 0), 则跳过进行下一次循环.

读取 X 矩阵的第 j 列, x_j 计算 $x_i x_j$

根据 S_{ij} 的符号, 加或者减, 累积到 M_l 矩阵上.

End

如果 M_l 不为全零矩阵, 且不是主节点, BSP 过程输出 M_l 矩阵到主节点.

主节点累加到 M_l 矩阵上, 得到最终的 M_l 矩阵.

算法结束.

M_l 计算过程, 最后在主节点上的累加过程, 需要进行 p 次 D*D 的矩阵加法运算. 通过二分累加的方法, 只需要 log p 次 D*D 的矩阵加法运算.

5 实验

实验在 CIFAR-10 数据集[10]上进行, 该数据集有 8 千万张小图片. 图片共分为 10 个类别. 原始数据集合中已经有所有图片的分组信息. 这里为了测试半监督哈希算法, 利用原始数据集合的分组信息, 随机获得两张为一组的图片, 根据已知的分组信息, 得到标记信息, 作为监督性信息. 对于图片的特征, 采用 GIST 特征[11]. 该特征为 960 维高维向量. GIST 算法采取 LEAR 的 C 实现库[12]. 该库图像的输入格式为 PPM 格式, CIFAR 数据集合图像格式为原始的矩阵数据. 根据 PPM 图像文件格式, 需要相应的程序进行转换. 根

© 中国科学院软件研究所 http://www.c-s-a.org.cn

据上面的讨论, 半监督哈希算法, 其并行实现的关键在于 M 调整相关矩阵的计算过程. 根据第 4 节的分析, HAMA 的算法对矩阵的存储格式有特殊的要求, 利用其相应的 SequenceFile 类, 进行相应的转换过程.

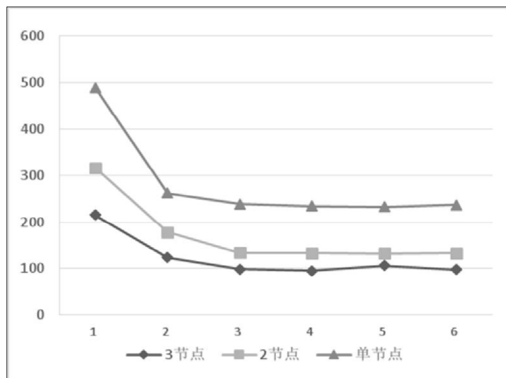

图 3 节点任务个数与运行时间的关系

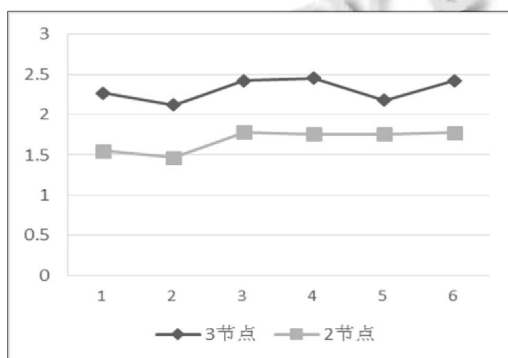

图 4 加速比与节点任务个数关系

实验环境为 3 节点的计算集群. 单机配置 Intel Core i5 4 核 CPU, 8G 内存, Linux 操作系统. 各节点上采用了不同的 Linux 发行版本, 不同机器上 JVM 版本也不完全一致, 6 或者 7. Hadoop 版本为 2.2.0, HAMA 为 0.7.0-SNAPSHOT 版本. 节点之间通过超五类网线和千兆交换机相连. 以上环境都是普通的服务器与网络环境, 因为我们考虑的就是在一般硬件水平下的情况.

在每个计算节点上, 需要 GroomServer 进行具体的计算过程, 需要 ZooKeeper 进行屏障同步. 因此一般计算节点上可以运行的最大 BSP 任务个数为 CPU 核数-1. 图 3 为计算节点任务个数不同情况下, 相同任务(这里计算任务为相关矩阵 M_C)所需要的时间. 可以看到每个节点上任务数为 3 时, 基本已经获得较好的性能, 而进一步提供单节点任务个数, 速度几乎没有变化. 甚至随着任务个数增加, 任务划分更细, 运行时间反而可能略长. 设定最大任务个数为 CPU 核数-1, 这样系统的容错性也更好. 因此后续的设置, 都是在单节点任务数为 3, 3 个计算节点的情况下进行. 类似的, 根据图 4, 加速比也是在节点个数为 CPU 核数-1 时较好.

根据图 4 也可以分析看到 2 节点相对于单个节点时候加速比为 1.77, 3 节点相对于单节点时候加速比为 2.43, 理论上完全并行的程序, 这里加速比应分别为 2 或者 3. 实验中略低是因为程序不可避免的串行部分, 以及节点增加带来的网络通信增加的开销. 这里验证 HAMA 平台水平扩展情况下的加速效果, 也就是增加机器个数可以有效的对任务进行并行加速.

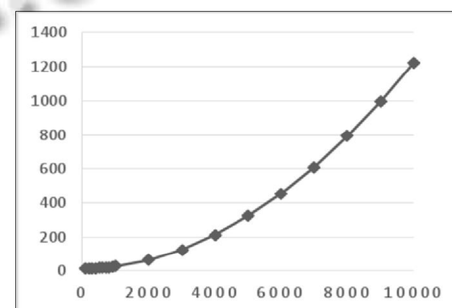

图 5 相关矩阵数据集与计算时间关系

对于相关矩阵的计算, 如图 5 可以看到随着测试数据集的个数增加, 所需要的时间也在随之增加. 值得注意的是所需要的时间, 并不仅仅是上面分析的线性增长. 这是因为 HAMA 本身框架就占用一些通信与计算的时间, 这些都会随着数据集合的增加而增加, 各个因素累加在一起的结果. 各个节点上需要在每个超级步时候进行同步, 因此速度就受限于最慢的计算任务. 对于标记矩阵, 取数据集为 10000, 随机选择标记信息的组数, 得到其性能曲线如图 6.

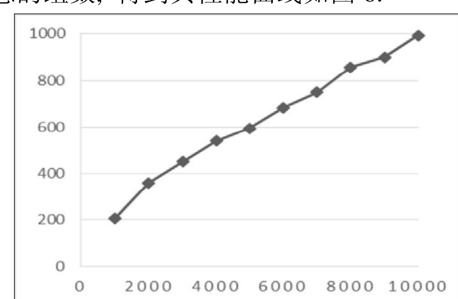

图 6 标记组数与计算时间关系

在取标记信息组数较少的时候, 标记信息比较分散, 载入特征向量相对其标记信息比较多, 因此在标

© 中国科学院软件研究所 http://www.c-s-a.org.cn

记组数较少的情况下，耗费的时间仍然较多.

得到调整矩阵 M 之后，可以利用单机的各种矩阵运算库. 实验中采用了 numpy 实现了后续的算法过程. 实验编写的部分源代码可以参见 https://github.com/liuyang1/ssh,https://github.com/liuyang1/hama.

6 结语

本文讨论了利用 HAMA 分布式框架，将半监督哈希算法分布式化. 特别是针对半监督哈希算法的调整相关矩阵的计算过程，进入了深入的分析. 完整实现了相关的半监督哈希算法的哈希函数的计算过程. 对于如何在实际中将一般算法，根据其特性，设计相应的分布式算法有一定的启发作用.

参考文献

1 Datta R, Joshi D, Li J, Wang JZ. Image retrieval: Ideas, influences, and trends of the new age. ACM Computing Surveys, 2008, 40(2): 1–60.

2 Muja M, Lowe DG. Fast Approximate nearest neighbors with automatic algorithm configuration. Proc. Int'l Conf. Computer Vision Theory and Applications. 2009. 331–340.

3 Wang J, Kumar S, Chang SF. Semi-supervised hashing for large-scale search. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2012, 34(12): 2392–2406.

4 Hadoop, http://hadoop.apache.org

5 陈国良.并行计算—结构算法编程,北京:高等教育出版社,2003.

6 M.F. Pace, BSP vs MapReduce, ICCS 2012.

7 HAMA: http://hama.apache.org

8 张韵华,奚梅成,陈效群.数值计算方法与算法.北京:科学出版社,2009:154–161.

9 ZooKeeper, http://zookeeper.apache.org

10 Torralba A, Fergus R, Freeman W. 80 Million tiny images: A large data set for nonparametric object and scene recognition. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2008, 30(11): 1958–1970.

11 Oliva A, Torralba A. Modeling the shape of the scene: A holistic representation of the spatial envelope. Int'l J. Computer Vision, 2001, 42(3): 145–175.

12 Douze M, Jégou H, Sandhawalia H, Amsaleg L, Schmid C. Evaluation of GIST descriptors for web-scale image search. International Conference on Image and Video Retrieval. July 2009.

13 Seo S, Yoon EJ, Kim J, Jin S, Kim JS, Maeng S. HAMA: An efficient matrix computation with the MapReduce framework. 2nd IEEE International Conference on Cloud Computing Technology and Science. 2010.

© 中国科学院软件研究所 http://www.c-s-a.org.cn