MapReduce 并行化压缩近邻算法

翟俊海[1,2],郝 璞[1],王婷婷[1],张明阳[1]

[1](河北大学 数学与信息科学学院 河北省机器学习与计算智能重点实验室,河北 保定 071002)
[2](浙江师范大学 数理与信息工程学院,浙江 金华 321004)
E-mail:mczjh@126.com

摘 要: 压缩近邻(CNN:Condensed Nearest Neighbors)是 Hart 针对 K-近邻(K-NN:K-Nearest Neighbors)提出的样例选择算法,目的是为了降低 K-NN 算法的内存需求和计算负担.但在最坏情况下,CNN 算法的计算时间复杂度为 $O(n^3)$,n 为训练集中包含的样例数.当 CNN 算法应用于大数据环境时,高计算时间复杂度会成为其应用的瓶颈.针对这一问题,本文提出了基于 MapReduce 并行化压缩近邻算法.在 Hadoop 环境下,编程实现了并行化的 CNN,并与原始的 CNN 算法在 6 个数据集上进行了实验比较.实验结果显示,本文提出的算法是行之有效的,能解决上述问题.

关 键 词: 压缩近邻;K-近邻;样例选择;MapReduce
中图分类号: TP181 **文献标识码:** A **文 章 编 号:** 1000-1220(2017)12-2678-05

Parallelization of Condensed Nearest Neighbor Algorithm with MapReduce

ZHAI Jun-hai[1,2], HAO Pu[1], WANG Ting-ting[1], ZHANG Ming-yang[1]

[1](Key Lab. of Machine Learning and Computational Intelligence, College of Mathematics and Information Science, Hebei University, Baoding 071002, China)
[2](College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, Jinhua 321004, China)

Abstract: CNN (Condensed Nearest Neighbors) proposed by Hart is an instance selection algorithm which aims at decreasing the memory and computation requirements. However, in the worst cases, the computational time complexity of CNN is $O(n^3)$, where, n is the number of instances in a training set. When CNN is applied to big data, high computational time complexity will become the bottleneck of its application. In order to deal with this problem, a parallelized CNN with MapReduce is proposed in this paper. We implement the proposed algorithm in Hadoop environment, and experimentally compare it with original CNN on 6 data sets. The experimental results show that the proposed algorithm is effective and efficient, and can overcome the mentioned problem.

Key words: condensed nearest neighbors; K-nearest neighbors; instance selection; MapReduce

1 引 言

K-近邻(K-NN:K-Nearest Neighbors)是著名的数据挖掘算法[1],2006 年在国际数据挖掘学术会议上被评为十大数据挖掘算法之一(名列第 8)[2].K-NN 算法虽然历史悠久,但由于其思想简单、易于实现、泛化能力强,K-NN 算法广泛应用于机器学习、数据挖掘和模式识别领域[3-5].K-NN 属于基于样例的学习算法,它不需要训练分类器模型,只需要存储所有的训练样例,用于计算待分类样例与训练样例之间的距离.K-NN 算法的计算时间复杂度和空间复杂度都是 $O(n)$,n 为训练集中包含的样例数.当 n 非常大时,K-NN 算法的效率会很低[6].为了加速 K-NN 算法的训练,有两种加速 K-NN 算法的策略:一是加速最近邻的计算,二是降低训练集的大小[7].

在第一种策略中,常用的加速方法是用近似近邻搜索代替精确近邻搜索[8].基于这种加速策略的方法又可以进一步划分为:基于树的方法和基于哈希技术的方法[9].基于树的方法利用树(如 KD-树[10]、VP-树[11]等)组织样例空间,并用基于树的快速搜索方法加速近邻搜索的速度.基于哈希技术的方法[12-14]将每一个样例变换为一个 0-1 串,而二进制计算能够加速近邻搜到的速度.在这类方法中,比较有影响的工作包括:Hou 等人提出的基于树的紧哈希近似近邻搜索方法[15],Slaney 和 Casey 提出的局部敏感性哈希近似近邻搜索方法[16].参考文献[17]对这类方法进行了全面的综述,具有较高的参考价值.

在第二种加速策略中,常用方法是从样例全集中选择一个重要的子集代替样例全集进行分类.CNN 是历史上第一个降低训练集大小的方法[18],CNN 算法从分类边界附近选择重要的样例,以降低训练集的大小.在近邻分类的框架中,从某种意义上说,后来的样例选择算法(例如,约简近邻算法[19]、编辑近邻算法[20]、迭代过滤算法[21]等)都是在 CNN 的基础上提出的.关于这类算法的性质和特点,有兴趣的读者可参考综述文献[6].

收稿日期:2016-10-24 收修改稿日期:2017-04-01 基金项目:国家自然科学基金项目(71371063)资助;河北省自然科学基金项目(F2017201026)资助;浙江省计算机科学与技术重中之重学科(浙江师范大学)课题项目资助. 作者简介:翟俊海,男,1964 年生,博士,教授,CCF 会员,研究方向为机器学习和数据挖掘;郝 璞,男,1992 年生,硕士研究生,研究方向为云计算与大数据处理;王婷婷,女,1991 年生,硕士研究生,研究方向为云计算与大数据处理;张明阳,男,1991 年生,硕士研究生,研究方向为云计算与大数据处理.

K-NN 算法的一个严重不足在于需要存储全部的训练样本,以及繁重的距离计算任务. CNN 算法正是为了降低 K-NN 算法的内存需求和计算负担而提出的,但是在最坏情况下,CNN 算法的计算时间复杂度为 $O(n^3)$. 当训练集非常大时(例如,训练集是大数据集),CNN 算法的应用就会非常困难. 针对这一问题,本文提出了 MapReduce 并行化压缩近邻算法,旨在通过 MapReduce 并行计算框架的优势,在海量数据中通过短时间的操作在原有样本集中挑选出对分类计算有效的样本,使样本总数合理的减少,同时达到既减少计算量,又减少存储量的双重效果.

2　基础知识

本节简要介绍将要用到的基础知识,包括 CNN 算法[18]和 MapReduce 编程模型[22].

2.1　CNN 算法

CNN 是 1967 年 Hart 针对 1 近邻(K = 1)提出的样例选择算法. 设 T 是训练集,T'是选择样例的集合. 初始时,从训练集中 T 中随机选择一个样例,加入到 T'中. 然后,递归地从训练集中选择样例. 每次都是随机地从 T 中选择一个样例,如果该样例被 T'中的样例用一近邻(K = 1)错误分类,则将其加入到 T'中. 否则,丢弃该样例. 直到下列条件之一满足,算法终止. 1)训练集为空;2)训练集中的样例都能被 T'中的样例正确分类. CNN 算法的伪代码如算法 1 所示.

算法 1. CNN 算法

```
输入:训练集 T = {(x_i, y_i) | x_i ∈ R^d, y_i ∈ Y, 1 ≤ i ≤ n}
输出:T' ⊆ T
初始化 T' = φ;
从 T 中随机地选择一个样例移动到 T'中;
repeat
    for(each x_i ∈ T) do
        for(each x_j ∈ T') do
            计算 x_i 到 x_j 之间的距离;
            //寻找 x_i 的 1-NN
            寻找 x_i 在 T'中的最近邻 x_j*;
        end
        if (x_i 的类别和 x_j* 的类别不同) then
            //x_i 不能被 T'用 1-NN 正确分类
            T' = T' ∪ {x_i};
            T' = T - {x_i};
        end
    end.
until(T = ∅或 T 中的所有样例都能被 T'用 1-NN 正确分类);
输出 T'.
```

如果 K > 1,CNN 算法只需初始化时,从训练集 T 中随机选择 K 个样例加入 T'中. 其他步骤不变.

说明:CNN 算法的核心概念是一致子集. 训练集 T 的子集 T'称为一致子集,如果 T'能够正确分类 T 中的所有样例. 训练集 T 的所有一致子集中,包含样例数最少的一致子集,称为 T 的最小一致子集. 实际上,CNN 算法试图寻找训练集 T 的最小一致子集,但该算法最终得到的子集 T'未必是最小一致子集.

2.2　MapReduce 编程模型

MapRecuce[22]是针对大数据处理的一种并行编程框架,它的基本思想包括以下 3 个方面:

1)MapRecuce 采用分治策略自动地将大数据集划分为若干子集,并将这些子集部署到不同的云计算节点上,并行地对数据子集进行处理;

2)基于函数编程语言 LISP 的思想,MapRecuce 提供了两个简单易行的并行编程方法:Map 和 Reduce. 用它们去实现基本的并行计算;

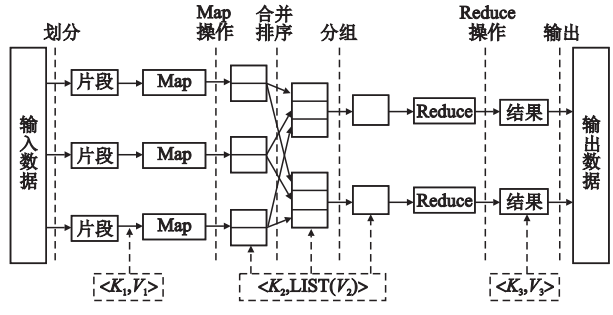

图 1　MapRecuce 处理数据的流程示意图

Fig. 1　Diagram of data processing by MapRecuce

3)许多系统级的处理细节 MapRecuce 能自动完成,这些细节包括:

a)计算任务的自动划分和自动部署;

b)自动分布式存储处理的数据;

c)处理数据和计算任务的同步;

d)对中间处理结果数据的自动聚集和重新划分;

e)云计算节点之间的通讯;

f)云计算节点之间的负载均衡和性能优化;

g)云计算节点的失效检查和回复.

MapRecuce 处理大数据的流程如图 1 所示.

3　MapReduce 并行化压缩近邻算法

本节介绍提出的 MapReduce 并行化压缩近邻算法,为描述方便,简记提出的算法为 MR-CNN. 对于给定的训练集 T 和初始候选样例子集 T'. 首先,算法 MR-CNN 从 T 中随机选择一个样例,加入到 T';然后,将 T 用 MapReduce 的 Map 机制划分为 m 个子集 $T_1, T_2, \cdots, T_m$,并部署到 m 个云计算节点上. 然后,在 m 个云计算节点,以 T'为初始候选样例子集(各个节点上全部一样的),应用 CNN 算法从相应的子集中选择样例,得到 m 个选择的样例子集 $T'_1, T'_2, \cdots, T'_m$. 其次,用 MapReduce 的 Reduce 机制将选择的 m 个选择的样例子集 $T'_1, T'_2, \cdots, T'_m$ 合并,得到运行一次 MapReduce 选择的样例子集 $T' = \cup_{i=1}^{m} T'_i$. 最后,用新的 T'作为初始候选样例子集,重复上述过程,直到 T'中的样例不再变化为止.

在 Hadoop 环境下,用 MapReduce 实现 CNN 算法,关键是 Map 函数和 Reduce 函数的设计,两个函数的设计如下:

Map 函数的具体设计如下:遍历所有训练样本点 x_i,计算它与 condenseSet 存储的数据集中样例之间的欧式距离,并求

出最近距离 minDistance 与相应最近样本 instance,判断 instance 的类别与 x_i 的类别是否相同,如果不同,将 x_i 加入到 condenseSet 数据集中,最后将压缩后的样本进行输出. 在 Map 函数中, < k1, v1 > 为 < 起始偏移量,训练样本 > ; < k2, v2 > 为 < 压缩样本, NullWritable > . 设计的 Map 函数的伪代码如算法 2 所示.

Reduce 函数的设计比较简单:遍历所有压缩后得到的样本点,并进行输出即可. 在 Reduce 函数中, < k2, v2 > 为 < 压缩样本, NullWritable > ; < k3, v3 > 为 < 压缩样本, NullWritable > . 设计的 Reduce 函数的伪代码如算法 3 所示.

算法 2. Map 函数

```
输入: <k1, v1>
输出: <k2, v2>
//遍历所有训练样本点 xi,计算其与 condenseSet 所有数据集之间的欧式距离,并求出最近距离 minDistance 与相应最近样本 instance
for(i=1; i≤n; i=i+1) do
    for(j=0; j<condenseSet.size(); j=j+1) do
        distance-EuclideanDistance(xi, condenseSet.get(j));
        if(distance≤minDistance) then
            minDistance = distance;
            instance = condenseSet.get(j);
        end
    end
end
//判断 instance 的类别与 xi 的类别是否相同,如果不同,将 xi 加入到 condenseSet 数据集中
if(instance.getLable() != xi.getLable()) then
    condenseSet.add(xi)
end
//将压缩后的样本进行输出
for(i=0; i<condenseSet.size(); i=i+1) do
    context.write(condenseSet.get(i), NullWritable.get());
end
输出 <k2, v2>
```

算法 3. Reduce 函数

```
输入: <k2, v2>
输出: <k3, v3>
//遍历所有压缩后得到的样本点,并进行输出
for(j=0; j<condenseSet.size(); j=j+1) do
    context.write(condenseSet.get(i), NullWritable.get());
end
输出 <k3, v3>.
```

4 实验结果与分析

为了验证本文提出的算法的有效性,我们在 6 个大数据集上进行了实验,并与 CNN 算法进行了实验比较. 选择的 6 个数据集包括 4 个 UCI 数据集和 2 个人工数据集. 众所周知, CNN 算法选择的样例大都分布在分类边界附近[18]. 因为人工数据集的分类边界是已知的,这样用人工数据集可以验证由本文算法选择的样例是否也在分类边界附近. 另外,选择人工数据集还可以验证本文算法的可行性.

第一个人工数据集是 3 类二维数据集,每类 100000 个样例,共 300000 个样例. 3 类服从的概率分布为:

$$p(\boldsymbol{x}|\omega_1) \sim N(\mathbf{0}, \mathbf{I}), p(\boldsymbol{x}|\omega_2) \sim N\left(\begin{pmatrix}1\\1\end{pmatrix}, \mathbf{I}\right),$$

$$p(\boldsymbol{x}|\omega_3) \sim \frac{1}{2}N\left(\begin{pmatrix}0.5\\0.5\end{pmatrix}, \mathbf{I}\right) + \frac{1}{2}N\left(\begin{pmatrix}-0.5\\0.5\end{pmatrix}, \mathbf{I}\right).$$

第二个人工数据集是一个 2 类二维数据集,每类包含 200000 个样例点,共 400000 个样例点. 两类服从的高斯分布为 $p(x|\omega_1) \sim N(\mu_i, \Sigma_i)$, $i=1,2$, 参数如下表所示.

i	μ_i	Σ_i
1	$\begin{pmatrix}1.0\\1.0\end{pmatrix}$	$\begin{pmatrix}0.6 & -0.2\\-0.2 & 0.6\end{pmatrix}$
2	$\begin{pmatrix}2.5\\2.5\end{pmatrix}$	$\begin{pmatrix}0.2 & -0.1\\-0.1 & 0.2\end{pmatrix}$

对于每一个大数据集,实验采用的方法是按 7:3 的比例随机地将其划分为训练集和测试集. 实验所用的 6 个大数据集的基本信息列于表 1 中. 我们将实验的源代码上传到了网站:http://pan.baidu.com/s/1cqkM3k,有兴趣的读者下载参考.

表 1 实验所用数据集的基本信息

Table 1 Basic information of data sets used in our experiments

数据集	样例个数	属性个数	类别个数
Gaussian1	300000	2	3
Gaussian2	400000	2	2
Statlog	43500	9	7
Skin Segmentation	240000	3	2
Poker Hand	1000000	10	10
Forest CoverType	580000	54	7

实验环境是在 Linux 环境下搭建的伪分布云计算平台,这些节点的基本配置信息和主机配置如下所示.

主机基本配置信息如表 2 所示.

表 2 主机基本配置信息

Table 2 Basic configuration of the host

软硬件项目	配置情况
处理器(CPU)	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
内存	4GB RDIMM
硬盘	1 TB
操作系统	Win7 64 位
程序开发环境	eclipse-jee-kepler-SR2-win32
JDK 版本	jdk_8u60_windows_i586_V8.0.600.27.1440040557

伪云计算平台节点的基本配置信息如表 3 所示.

我们在能力保持的前提下,将本文算法 MR-CNN 与 CNN 在运行时间上进行了比较,实验结果列于表 4 中. 实际上,用本文算法选择的样例也都分布在分类边界附近,由于篇幅所限,这里不再给出直观的示意图. 从表 4 可以看出,本文算法 MR-CNN 在运行时间上远远低于 CNN 算法. 而且,在 Poker Hand 和 Forest CoverType 这两个数据集上,CNN 算法得不到运行结果(表中用"--"表示),而本文算法能够得到结果,而且运行速度也比较快. 从列于表 4 的实验结果可以看

出,本文提出的算法是很有效的.

表 3　伪云计算平台节点的基本配置信息

Table 3　Basic configuration of the nodes of the pseudo cloud computing platform

软硬件项目	配置情况
处理器(CPU)	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
内存	1GB RDIMM
硬盘	8GB
操作系统	Linux
程序开发环境	Hadoop-1.1.2
JDK 版本	jdk_8u60_windows_i586_V8.0.600.27.1440040557

需要进一步说明的是,为了比较的公平,我们在单机环境下,用伪云计算平台进行对比实验. 如果在真正的集群环境下,根据 MapReduce 的工作机制原理,在多个节点并行操作过程中,速度会有更大提升. 单机环境下的 CNN 只能在一个节点上工作,无法与其他节点进行合作处理数据,面对大数据集需要花费大量时间处理,甚至会导致内存不足程序出错;而基于 MapReduce 的 CNN 算法可以加入任意数量的节点,不会出现内存不足的问题,并且有并行处理的优势. 在实验中,类似数据集 Poker Hand,传统 CNN 算法所需时间内无法完成;但是基于 MapReduce 的 CNN 就可以运行成功.

MapReduce 运行中利用 Namenode 对集群进行分片,就是一个 split,一个 split 对应一个 Map 任务,而每一个节点的通信都是需要时间的,也需要耗费一定的资源,而且在单机下只有一个 CPU,任务只能顺序执行,所以任务越多,时间运行越长,任务计算部分所要耗费的时间,掩盖了 MapReduce 并行计算的优势,这也是为什么很多文章都提到的 MapReduce 框架不适用于小数据文件原因.

MapReduce 拥有集群优势,MR-CNN 算法的运行速度是单机版 CNN 的 3 到 8 倍,随着节点数的增加和数据集大小的增加,这个优势会更加明显.

表 4　MR-CNN 与 CNN 比较的实验结果(单位:秒)

Table 4　Comparison of experimental results of MR-CNN and CNN

数据集	样例个数	CNN	MR-CNN
Gaussian1	200000	16207	2697
Gaussian2	400000	74968	34085
Statlog	43500	91	16
Skin Segmentation	240000	29	10
Poker Hand	1000000	--	69812
Forest CoverType	580000	--	50498

为了进一步验证 MR-CNN 的有效性,本文随后在 6 台服务器构成的的云平台上面进行了实验,实验证明本文提出的想法是行之有效的. 云平台的基本配置信息如表 5 所示.

节点的具体规划如表 6 所示.

5　结　论

在大数据环境,针对 CNN 算法的不足,提出了一种 MapReduce 并行化 CNN 算法,可显著提高 CNN 算法运算速度,可实现对大数据集的压缩. 提出的算法具有如下两个特点:

表 5　云计算平台节点的基本配置信息

Table 5　Basic configuration of the nodes of the cloud computing platform

软硬件项目	配置情况
处理器(CPU)	Inter Xeon E5-4603 2.0GHz (双核)
内存	8GB RDIMM
硬盘	1 TB
网卡	Broadcom 5720 QP 1Gb 网络子卡(四端口)
网络设备	华为 S3700 系列以太网交换机
操作系统	Ubuntu kylin 13.04
云计算平台	Hadoop0.20.2
JDK 版本	Jdk-7u71-linux-i586
程序开发环境	Eclipse-Java-luna-SRI-linux

1)算法思想简单,易于编程实现;

2)算法运行效率高,所用的运行时间远远低于 CNN 所用的运行时间.

3)Hadoop MapReduce 在处理循环时由于需要大量的磁盘读写操作,确实效率很低.

表 6　云计算平台节点功能规划

Table 6　Role assignment of the nodes of the cloud computing platform

节点号	主机名	IP 地址	节点类型
1	hadoop1	10.187.84.50	NameNode,SecondaryNameNode
2	hadoop2	10.187.84.51	JobTracker
3	hadoop3	10.187.84.52	DataNode,TaskTracker
4	hadoop4	10.187.84.53	DataNode,TaskTracker
5	hadoop5	10.187.84.54	DataNode,TaskTracker
6	hadoop6	10.187.84.55	DataNode,TaskTracker

但是本文提出的算法在实际操作的过程中发现循环操作只需 2 到 3 次就可以满足实验要求,时间消耗不是太大,因此本文提出的 MapReduce 并行化 CNN 算法更关注于海量数据的处理过程. 未来进一步的工作包括:

1)在更多、更大的数据集上实验;

2)与更多的相关方法进行实验比较;

3)对现有算法提出创新改进,优化效率更高;

4)对实验结果进行深入的分析,包括对实验结果的统计分析.

References:

[1] Cover T, Hart P. Nearest neighbor pattern classification[J]. IEEE Transactions on Information Theory, 1967, 13(1): 21-27.

[2] Wu X, Kumar V, Quinlan J R, et al. Top 10 algorithms in data mining[J]. Knowledge & Information Systems, 2007, 14(1): 1-37.

[3] Muja M, Lowe D G. Scalable nearest neighbor algorithms for high dimensional data[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2014, 36(11): 2227-2240.

[4] Dasarathy B. Nearest neighbor (NN) Norms: NN pattern classification techniques [M]. Vlos Alamitos: IEEE Computer Society Press, 1991.

[5] Sarkar M. Fuzzy-rough nearest neighbor algorithms in classification [J]. Fuzzy Sets and Systems, 2007, 158(19): 2134-2152.

[6] Salvador G, Joaquin D, Jose R C, et al. Prototype selection for nearest neighbor classification: taxonomy and empirical study [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012, 34(3): 417-435.
[7] Alvar A G, Jose-Francisco D P, Rodríguez J J, et al. Instance selection of linear complexity for big data [J]. Knowledge-Based Systems, 2016, 107: 83-95.
[8] Andoni A, Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions [J]. Communications of the ACM, 2008, 51 (1): 117-122.
[9] Liu S G, Wei Y W. Fast nearest neighbor searching based on improved VP-tree [J]. Pattern Recognition Letters, 2015, 60-61: 8-15.
[10] Herranz J, Nin J, Sole M. KD-trees and the real disclosure risks of large statistical databases [J]. Information Fusion, 2012, 13(4): 260-273.
[11] Yianilos P N. Data structures and algorithms for nearest neighbor search in general metric spaces [C]. Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, Texas, USA. January 25-27, 1993: 311-321.
[12] Li Wu-jun, Zhou Zhi-hua. Learning to hash for big data: current status and future trends [J]. Chinese Science Bulletin, 2015, 60(5): 485-490.
[13] Wang Jian-feng. Hashing-based nearest neighbot search [D]. Hefei: University of Science and Technology of China, 2015.
[14] Chang C C, Wu T C. A hashing-oriented nearest neighbor searching scheme [J]. Pattern Recognition Letter, 1993, 14(8): 625-630.
[15] Hou G, Cui R, Pan Z, et al. Tree-based compact hashing for approximate nearest neighbor search [J]. Neurocomputing, 2015, 166: 271-281.
[16] Slaney M, Casey M. Locality-sensitive hashing for finding nearest neighbors [J]. IEEE Signal Processing Magazine, 2008, 25(1): 128-131.
[17] Pauleve L, Jegou H, Amsaleg L. Locality sensitive hashing: a comparison of hash function types and querying mechanisms [J]. Pattern Recognition Letters, 2010, 31(11): 1348-1358.
[18] Hart P E. The condensed nearest neighbor rule [J]. IEEE Transaction on Information Theory, 1968, 14(5): 515-516.
[19] Gates G W. The reduced nearest neighbor rule [J]. IEEE Transactions on Information Theory, 1972, 18(3): 431-433.
[20] Wilson D R, Martinez T R. Reduction techniques for instance-based learning algorithms [J]. Machine Learning, 2000, 38(3): 257-286.
[21] Brighton B, Mellish C. Advances in instance selection for instance-based learning algorithms [J]. Data Mining and Knowledge Discovery, 2002, 6(2): 153-172.
[22] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

附中文参考文献：

[12] 李武军，周志华. 大数据哈希学习：现状与趋势 [J]. 科学通报，2015，60(5)：485-490.
[13] 王建峰. 基于哈希的最近邻查找 [D]. 合肥：中国科学技术大学，2015.