

FP-Growth 算法 MapReduce 化研究

吕雪骥, 李龙澍

(安徽大学 计算机科学与技术学院, 安徽 合肥 230601)

摘要:随着云计算概念的盛行,以及数据挖掘技术在分布式环境下的应用问题,该文献针对当前业界中流行的大规模并行计算模型 MapReduce,将其引入数据挖掘领域关联规则算法的并行化改进中,提出基于 FP-Growth 算法并行化改进的 MR-FP 算法,为并行化关联规则挖掘提供节点可扩展、可容错、故障可恢复的运行保证。并通过案例分析得出系统在事务数呈数量级增长下仍可保持较高的性能。通过理论分析和案例实验表明,数据挖掘理论和方法在云计算环境下可以充分发挥能力,具有广阔的、有价值的研究空间。

关键词:MapReduce; FP-Growth; MR-FP; 云计算; 分布式数据挖掘

中图分类号: TP311

文献标识码: A

文章编号: 1673-629X(2012)11-0123-04

Research on Improved FP-Growth Algorithm with MapReduce

LÜ Xue-ji, LI Long-shu

(College of Computer Science and Technology, Anhui University, Hefei 230601, China)

Abstract: Nowadays, the massively parallel computing model MapReduce is very popular in the current industry. It will introduce it into the improvement of association rules of data mining algorithms in parallelization, propose the improved MR-FP algorithm based on paralleled FP-Growth algorithm, and provide the parallelization association rules mining with node scalable, fault tolerance and operation. Draw a conclusion that the system can still maintain pretty high performance when the transactions are under the orders of magnitude level. The theoretical analysis and the case studies demonstrate that data mining theory and methods can show their full abilities based on the cloud computing. It deserves more valuable research.

Key words: MapReduce; FP-Growth; MR-FP; cloud computing; distributed data mining

0 引言

随着计算机技术的高速发展,尤其是近年来云计算概念在业界的热议,并行计算和分布式计算技术在数据挖掘学术界得到广泛关注,并行、分布式的以及在云计算下的数据挖掘成为当前研究的热点之一。

Google 公司的 MapReduce 编程模型为并行和分布式计算提出了一种简单的模型,并且该模型被广泛运用到当今业界的云计算核心技术中。

文中将关联规则挖掘中的 FP-Growth 算法经过 MapReduce 编程模型的改造,使之适应于 MapReduce 框架下的并行计算,达到 FP-Growth 算法的并行化目的,同时该算法也将达到在云计算中应用的可行性。

1 相关概念

1) FP-Growth。

FP-Growth^[1]算法是由 Han Jiawei 等提出的一种基于频繁项集的关联规则挖掘算法。其算法核心是通过两趟数据库扫描构造 FP 树 (FP-Tree) 和头表 (Header Table),从而得到用于频繁项集挖掘的压缩的数据库映射。之后对每个频繁项构造其条件 FP 树进行频繁项集的挖掘。

FP-Growth 算法和传统的 Apriori 算法比较具有以下优点:

- (1) 仅 2 次扫描数据库;
- (2) 不生成候选项集;
- (3) 事务数据在内存中以压缩的 FP 树方式表示。

以上 3 点优势,使得 FP-Growth 算法在 Apriori 算法基础上有数量级级别的性能提升。

但是通过一味提高算法的性能是有限的,随着当今业界的飞速发展,即使是高性能的计算机亦无法达到在用户可接受的响应时间内完成分析当今如此海量数据的能力。FP-Growth 算法在关联规则挖掘中随着

收稿日期: 2012-03-05; 修回日期: 2012-06-10

基金项目: 安徽省自然科学基金 (090412054)

作者简介: 吕雪骥 (1987-), 男, 硕士研究生, 研究方向为数据挖掘技术; 李龙澍, 教授, 硕士生导师, 研究方向为智能软件、机器人足球、粗糙集理论、数据挖掘。

事务数据规模的急剧增大,也会出现性能问题。由于 FP-Growth 算法将事务数据中的频繁部分在内存中压缩成 FP 树,当数据规模很大时,FP 树规模会相当可观。算法中对 FP 树的递归构造也会大大降低运算性能。因此自然会思考到通过并行、分布式的计算方式来提高运算效率。对于并行的关联规则挖掘,Agarwal 等在文献[2]中提出了 CD、CaD 和 DD 算法。曾志勇等在文献[3]中提出了一种负载均衡的并行 FP-Growth 算法改进,通过多节点均衡地参与计算,达到降低算法执行时间的目的。文献[4]中提出了一种基于 Web Service 的分布异构环境下的数据挖掘框架,并将改进的 FP-Growth 算法应用到框架之中。然而随着事务规模的增大,必然需要增加节点数量,当前经典的并行计算方法对于节点数量的扩展不具有很好地支持;同时算法在实际生产环境中还需要考虑多节点所带来的节点故障、网络通信故障等复杂问题。

2) MapReduce.

MapReduce^[5]是由 Google 公司提出的一种并行程序设计的编程模型。MapReduce 编程模型的原理就是利用一个输入的键值对集合来产生一个输出的键值对集合。用户需要定义两个函数:map 和 reduce, map 函数接收输入的键值对分片,产生中间键值对集合。MapReduce 将键相同的键值对组合在一起合并到 map 函数所在节点的磁盘上,然后将这些不同键的集合分别发送到不同的 reduce 节点上。reduce 节点接收了多个 map 节点发送来键值对集合,将它们通过 reduce 函数合并成较小的集合,大量的中间键值对通过迭代调用 reduce 函数,最后生成结果。该模型将并行程序设计思想体现在其核心概念“Map”和“Reduce”上。简单地说,MapReduce 通过 Map 操作和 Reduce 操作建立其算法的输入到算法输出之间的桥梁。Map 实现输出数据分发,将算法输入经过各个节点处理后分发到各个 Reduce 节点;Reduce 实现核心算法并输出结果^[6,7]。

如图 1 所示,在 MapReduce 过程中需要有一个主节点(Master),它存储着所有 map 和 reduce 的工作状态以及系统中的节点标识。主节点为系统中的节点分配 map 和 reduce 任务,并监控着它们完成各自的工作。当某个 map 节点工作完成后,该节点产生的中间结果将会存储在本节点的磁盘内,并把结果的大小和位置发送给主节点。主节点会逐步将需要这些中间结果的 reduce 节点告知中间结果的存储情况,由 reduce 从 map 节点的磁盘上读取。

MapReduce 的编程模型具有很高的容错性,当一个 map 节点或 reduce 节点出现故障时,主节点收不到该节点的回应,认为节点出现故障,将该节点排除在系

统之外。节点上所处理的键值对将会重新分配到其它 map 或 reduce 节点上重新执行。待故障节点的故障被排除,主节点重新再给该节点安排新任务。主节点上的数据周期性地做为检查点(checkpoint)被保存在本地磁盘上,当主节点发生故障可从检查点恢复工作。

在集群上运行 MapReduce 程序的时候,开发者并不需要关心输入的数据是如何分配到各个节点上的,以及节点之间数据和任务的调度,节点故障后该节点上的数据和计算任务如何重新分配到其它节点重新计算等等具体的分布式计算中的问题。

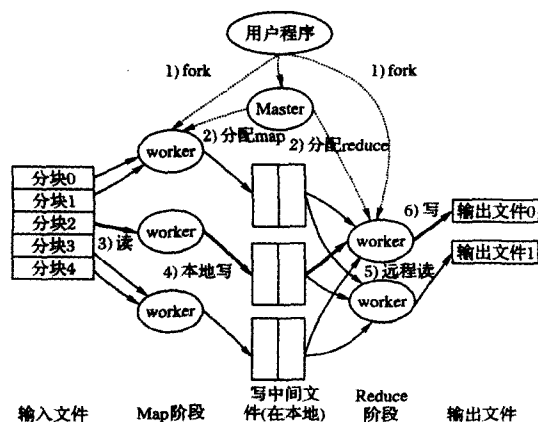


图 1 MapReduce 过程

采用 MapReduce 的优势在于,通过简单的编程模型达到分布式计算的目的,同时使得程序具有很强的扩展性、容错性、故障恢复能力。

多位学者对基于 MapReduce 的数据挖掘算法改进进行了研究。文献[8]在 MapReduce 框架下对多种机器学习算法进行了改进和研究。文献[9,10]提出了基于 Apriori 算法的 MapReduce 化改进方法。然而文中选择对 FP-Growth 算法进行 MapReduce 化改进,原因在于:一来 FP-Growth 算法具有更高的性能和更低的内存开销,关于这点文献[11]中进行了阐述;二来 Apriori 算法在计算频繁项集时每一阶的频繁项集迭代均需要从各个节点合并局部解,这带来很大的额外开销。而 FP-Growth 算法求频繁项集不是按阶计算而是按频繁项计算,因此只要保证关于一个频繁项的事务在一个计算节点上可以找到,则关于该频繁项在该节点上形成闭包,对于频繁项集的计算无需和其它节点通信,提高了分布式计算效率。

2 基于 MapReduce 的 FP-Growth 算法

在介绍基于 MapReduce 的 FP-Growth 算法(MR-FP 算法)之前,看一下传统 FP-Growth 算法的步骤:

(1)通过扫描一次事务数据库得到频繁项及其支持度计数,并对频繁项按其支持度计数从高到低进行排序。并构建头表(Head Table)。

(2)再次扫描事务数据库,将读取的每条事务按照第一步的频繁项顺序进行排序,排序后以 null 为根结点构建一条 FP 树的路径,对路径上项的计数加 1。在插入 FP 树过程中查找头表里对应的项,建立链表索引。

(3)从头表尾部向上遍历频繁项集,每次循环过程中从头表的链表里访问 FP 树得到条件模式,根据条件模式构建每个频繁项的条件 FP 树。

(4)从条件 FP 树分单分支和多分支两种情况递归挖掘频繁项集。

基于 MapReduce 的 FP-Growth 算法 MR-FP 和经典 FP-Growth 算法对应地具有以下两个步骤:

1)通过一次 MR 任务统计事务项频繁度。FP-Growth 算法在构造 FP 树之前需遍历一趟数据库统计事务项的频繁度并排序。这对应于 FP-Growth 算法的第一步。在 MapReduce 框架下,这个步骤可以通过一个 MapReduce 任务来分布式地计算完成。

2)通过一次 MR 任务执行 MR-FP 算法计算频繁项集。在 MR-FP 算法中 FP-Growth 算法的 2、3、4 步经过改进和并行化由一步 Map-Reduce 过程完成。这是算法的核心任务,通过本次 MR 任务将 FP-Growth 算法中对各频繁项的条件 FP 树挖掘分发到各个节点进行本地 FP-Growth 的第 4 步计算,再合并最终解。根据频繁项集得到最终的关联规则结果。

2.1 统计事务项频繁度

Map 输入:<null,事务>

Map 处理:

foreach I in Ti:

Output <I, 1>

Map 输出:<Ii,1>

Reduce 输入:<Ii,1>

Reduce 处理:

cnt = 0

while values.hasNext():

cnt = cnt + 1

Output<I, cnt>

Reduce 输出:<Ii,Ni>

在 Map 阶段 mapper 将输入的事务做为以 null 为键,事务为值的键值对,按项分散拆分成以事务中的项为键,1 为值的中间结果键值对,并通过 MapReduce 的 shuffle 按照键排序发送给 reducer,MapReduce 保证相同键的键值对由一个 reducer 节点接收。在 Reduce 阶段一个 reducer 将自己收到的键值对按键相同的累加得到一个键的统计计数。即一个项对应的频繁度。

2.2 计算频繁项集并收集

计算频繁项集在经典的 FP-Growth 算法中是其核

心所在。FP-Growth 算法通过构造 FP 树映射事务数据,然后从 FP 树中构造各频繁项的条件 FP 树,最后从条件 FP 树中得到包含该频繁项的频繁项集。因此,FP 树是从事务集到各频繁项的条件 FP 树的桥梁。然而在 MR-FP 算法中,无需构造整个事务集的 FP 树,仅需要将构造该频繁项条件 FP 树的事务发送到其计算节点上,由该节点构造该频繁项的条件 FP 树,并使用经典 FP-Growth 算法得到包含该频繁项的频繁项集并输出。

Map 输入:<null, Ti>

Map 输出:<Ii, t[0:curr-1]>

Map 处理:

L = GetSortedItemList()

t = Sort(Ti, L)

for I in t:

If Support(I) < minSup:

break

Output <I, t[0:curr-1]>

Reduce 输入:<Ii, t[0:curr-1]>

Reduce 输出:<null, FP>

Reduce 处理:

InitCFPTree(Tree)

while values.hasNext():

CFPTree_insert(Tree, values.next().get())

FP = LocalFPGrowth(Ii, Tree)

Output <null, FP>

在 Map 过程中,Map 输入为以 null 为键,事务记录为值的键值对。先得到上一步骤计算出来的频繁项排序数组,并将事务按频繁项频繁次数排序。然后遍历事务中的频繁项,将构造该频繁项所需的事务片段以频繁项为键,事务片段为值的键值对存储在磁盘上。在 Reduce 过程中,Reduce 节点收到主节点分配的 reduce 任务,并得到 map 节点计算的中间结果存放的位置和大小,从 map 节点磁盘上读取中间键值对做为输入,为频繁项对应的各个事务片段。将遍历这些事务片段,将它们构建起该频繁项的条件 FP 树。从条件 FP 树进而得到包含本频繁项的频繁项集,以 null 为键,频繁项集为值输出。最后由一个节点收集所有的频繁项集,并由此得到关联规则。

算法抛除了构造 FP 树和 head 表的过程,这为系统节省了很大的运算时间和内存占用。传统算法中的这个过程可以对事务进行压缩,降低扫描数据库次数。在分布式环境下的 MR-FP 算法中,对事务库的扫描也只有两次,因为虽然没有 FP 树,事务在 map 过程后已经被化简成以频繁项为键,事务片段为值的键值对,

分散存储在 map 节点的磁盘上了;而 head 表在 MR-FP 算法中频繁项以键的形式得以被索引,更能提高其检索效率。

3 案例分析

为了验证 MR-FP 算法大规模的事务关联规则挖掘上的优越性,采用 <http://cdiac.ornl.gov/ftp/> 上的气象数据进行离散化处理 after 做为事务数据源。通过关联规则挖掘,找寻气象要素之间的关联规则关系。

实验环境采用 4 台 PC 做为一个分布式环境。计算机操作系统为 openSUSE Linux 11.4, CPU 为 Inter Core i3 处理器,主频 3.2GHz,内存 4G。

在其中一台 PC 上运行 FP-Growth 算法,然后在 4 台 PC 上搭建 MapReduce 模型实现的框架 Hadoop^[12],运行 MR-FP 算法。运行效果统计如图 2 所示:

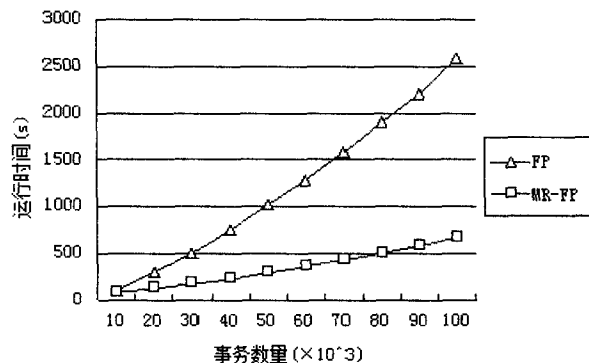


图2 MR-FP 算法与 FP-Growth 算法性能比较

从图1中可以看到,采用分布式算法 MR-FP 的运行效率对比于传统算法有很大提高,同时随着事务规模的增大,这种优势更趋于明显。由于运行 FP-Growth 算法仅使用了一台 PC,而 MR-FP 算法中使用了 4 台 PC,其中 1 台做为中央节点,负责任务的分配和分布式文件系统管理;另外 3 台做为计算节点,负责数据的存储和计算。因此在比较两个算法的优劣时需要考虑 FP-Growth 算法的运行时间平均到 3 台计算节点上的时间,以及平均到 4 台参与运算的节点上的时间。将 FP-Growth 算法的统计结果分别除 3 和除 4 后再与 MR-FP 算法统计结果比较,如图 3 所示。

从图2中可以看到,MR-FP 算法在事务规模较小时,平均效率低于传统 FP-Growth 算法。这是由于分布式计算中对于节点的管理和分配增加了额外的计算开销,在事务规模小时这个开销占了主要运行时间。而随着事务规模的增大,MR-FP 算法效率超过了 FP/3,趋近于 FP/4。这说明 MR-FP 算法在 3 台 PC 上计算的效率高于在 FP-Growth 算法在一台 PC 上的 3 倍,而 MR-FP 算法整个 MapReduce 环境所处的 4 台 PC 上计算的效率逐渐接近 FP-Growth 算法在一台 PC 上

的 4 倍。前者说明,MR-FP 算法本身效率优于 FP-Growth 算法;后者说明随着事务规模增大,分布式算法的节点管理和分配开销可以忽略不计。

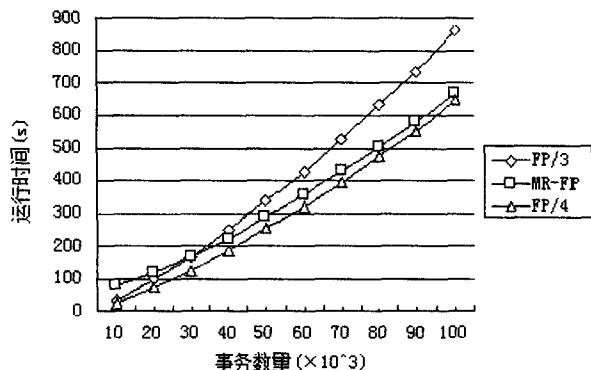


图3 MR-FP 算法与 FP-Growth 算法开销均分到 3 台和 4 台计算机上的性能比较

通过案例实验和结果分析可以对 MR-FP 算法的特点分析如下:

MR-FP 算法更适用于大规模集群和海量数据的关联规则挖掘。在实验中由于实验条件有限,集群规模和数据规模均远小于这个要求,但从中仍然可以看到算法效率和节点数量、数据规模呈正比的趋势。这是因为 MR-FP 将 FP-Growth 算法按照 MapReduce 编程模型进行了合理的 map 和 reduce 函数式划分,充分解除了求解条件 FP 树之间的计算耦合,达到并行化计算的目的。

随着数据规模的增大,传统算法的运算效率短板在于无法提供足够大小的内存空间。程序需要进行大量的外存 I/O 操作,且进行 I/O 操作时程序处于阻塞等待状态,这种情况在大规模的数据量下表现尤为明显,从而大大降低了算法效率。而 MR-FP 算法所处的分布式计算环境中,MapReduce 对外存数据以顺序文件的方式存储,读取数据的同时数据流就迭代地进行相应的 map/reduce 操作,而写入数据采用追加而非改写的方式。这种 I/O 方式大大降低了读写时间,也提高了算法的并行程度。

4 结束语

通过实验,达到了算法设计的预期。实验建立了一个高效、可扩展、高可用性的分布式关联规则分析系统。该系统在单节点上算法效率高于单机的 FP-Growth 算法;由于采用了 MapReduce 编程,系统通过简单地增加节点方式即可以提高并行程度、缩短运行时间;在 Hadoop 框架下,系统对于节点具有高容错性和高恢复性。在云计算的海量数据下,数据挖掘算法可以充分发挥它强大的能力。在以后的研究中,可以

(下转第 130 页)

WSO_N 的拓扑是由地形、信道特征和基于节点传输功率限制的某种链路决定协议来确定的。为保证较好的网络连接度,往往采用拓扑控制机制^[13],根据应用环境调整节点的传输范围,使得节点稀疏的区域获得可以接受的连接度,同时能够将节点密集地区的业务流量控制在合理的范围内,尽量减少网络拥塞。拓扑控制往往借助于功率控制机制来实现,目标是在网络连通度约束下最小化成本指标(如总功率)或者在传输功率约束下最大化网络吞吐量。

2 结束语

无线自组网的 QoS 体系结构涉及大量有待深入研究的问题。无线自组网通过使用分布式控制算法来组织和维护网络,网络健壮性和灵活性较好,但引入较大控制开销并消耗大量能量。为此,FQAW 采用分簇网络结构来减少控制开销,提高了网络的可扩展性和服务质量保障水平。此外,FQAW 的设计必须在综合考虑多种因素的基础上进行合理折衷,包括业务量特征、移动模式、硬件特性和信道条件等。分析模型只能研究特定的网络协议的行为,并且有时根本无法实现,为此,评价各部件的性能及其交互作用通常需要借助于模拟试验的方法,考察各个功能部件相对于节点移动性和传输范围等参数的敏感程度,从而可以评价协议的性能和估计各个部件的开销,以此来指导实际的无线自组网的设计。

参考文献:

- [1] 王海涛,朱震宇,付 鹰. 应急通信网络设计及关键技术探

(上接第 126 页)

考虑将其它数据挖掘算法通过 MapReduce 模型的改造,应用到 Hadoop 平台上。

参考文献:

- [1] Han Jiawei, Pei Jian, Yin Yiwen. Mining frequent patterns without candidate generation[C]//SIGMOD'00. [s. l.]: [s. n.], 2000.
- [2] Agrawal R, Shafer J C. Parallel mining of association rules [J]. IEEE Transactions on knowledge and data engineering, 1996, 8(6): 962-969.
- [3] 曾志勇,杨呈智,陶 冶. 负载均衡的 FP-Growth 并行算法研究[J]. 计算机工程与应用, 2010(4): 125-126.
- [4] 尉立伟,姜 浩,蒲安建. Web Service 架构下的分布式关联规则挖掘研究[J]. 计算机技术与发展, 2009, 19(4): 31-34.
- [5] Jeffrey D, Sanjay G. MapReduce: Simplified Data Processing on Large Clusters[J]. Commun. ACM, 2008, 51(1): 107-

讨[J]. 指挥信息系统与技术, 2010, 1(5): 28-33.

- [2] 郑少仁,王海涛,赵志峰,等. Ad hoc 网络技术[M]. 北京: 人民邮电出版社, 2005.
 - [3] 王 雪. 无线传感网络测量系统[M]. 北京: 机械工业出版社, 2008.
 - [4] Liu Tehuang, Liao Wanjiun. On Routing in Multichannel Wireless Mesh Networks: Challenges and Solutions[J]. IEEE Network, 2010, 22(1): 13-18.
 - [5] 王海涛,郑少仁,宋丽华. Ad hoc 网络中 QoS 保障机制的研究[J]. 通信学报, 2002, 23(10): 114-121.
 - [6] 徐昌彪. IP QoS 体系结构综述[J]. 重庆邮电学院学报, 2001, 13(2): 36-42.
 - [7] 王海涛,郑少仁. 移动 Ad hoc 网络中的分簇算法[J]. 解放军理工大学学报, 2004, 5(3): 28-32.
 - [8] 王海涛,刘晓明. Ad hoc 网络中跨层设计方法的研究[J]. 电信科学, 2005, 21(2): 22-26.
 - [9] Chen Jie, Lü Tiejun, Zheng Haitao. Joint cross-layer design for wireless QoS content delivery [C]//IEEE International Conference on Communications. Piscataway: IEEE Press, 2004: 4243-4247.
 - [10] 米志超. Ad hoc 网络的路由协议及 QoS 路由选择算法的研究[D]. 南京: 解放军理工大学, 2002.
 - [11] Lin C R. Real-time Support in Multihop Wireless Networks [J]. ACM Journal of Wireless Networks, 1999, 12(3): 125-135.
 - [12] Mohapatra P, Li Jian, Gui Chao. QoS in Mobile Ad hoc Networks[J]. IEEE Wireless Communications, 2003, 11(6): 115-120.
 - [13] Goldsmith A, Wicker S. Design Challenges for Energy-constrained Ad Hoc Wireless Networks[J]. IEEE Wireless Communications, 2002, 9(4): 8-27.
-
- [6] 谢桂兰,罗省贤. 基于 Hadoop MapReduce 模型的应用研究[J]. 微型机与应用, 2010(8): 4-7.
 - [7] 江务学,张 璟,王志明. MapReduce 并行编程架构模型研究[J]. 微电子学与计算机, 2011(6): 168-170.
 - [8] Chu C, Kim S, Lin Y, et al. Map-Reduce for machine learning on multicore [C]//NIPS'06. Cambridge, MA: MIT Press, 2006.
 - [9] 余楚礼,肖迎元,尹 波. 一种基于 Hadoop 的并行关联规则算法[J]. 天津理工大学学报, 2011(2): 25-28.
 - [10] 戎 翔,李玲娟. 基于 MapReduce 的频繁项集挖掘方法[J]. 西安邮电学院学报, 2011(7): 37-39.
 - [11] 何中胜,庄燕滨. 基于 Apriori & Fp-growth 的频繁项集发现算法[J]. 计算机技术与发展, 2008, 18(7): 45-52.
 - [12] Hadoop. The Apache Software Foundation[EB/OL]. 2010. <http://hadoop.apache.org/>.