基于MapReduce模型的排序算法优化研究

金 菁
(北京理工大学软件学院 北京 100081)

摘 要 MapReduce已经发展成为大数据领域标准的并行计算模型。理想情况下,一个MapReduce系统应该使参与计算的所有节点高度负载均衡,并且最小化空间使用率、CPU和I/O的使用时长以及网络传输开销。传统的算法往往只针对上述指标中的一种进行优化。在保持算法良好并行性基础上,对多个指标同时进行优化,提出了MapReduce优化算法的设计规范。针对数据处理领域最重要的排序算法进行理论分析,给出了多指标约束下的最后算法,并证明了该优化算法满足MapReduce优化算法规范。最后通过实验验证了优化的排序算法的有效性和效率。

关键词 MapReduce模型,优化算法,大数据,排序算法
中图法分类号 TP319 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.12.033

Research on Optimization of Sorting Algorithm under MapReduce

JIN Jing
(School of Software, Beijing Institute of Technology, Beijing 100081, China)

Abstract This specification is set for the theses to be published in computer applications and software, including fonts, margins, page size and print area. MapReduce is the standard parallel computing model on big data analysis. Ideally, a MapReduce system should make all nodes involved in computation highly load balancing, and minimize space usage, CPU, disk and network overhead. In fact, these principles are the MapReduce algorithm design guidelines. However, few studies on optimization achieve all these metrics simultaneously. To solve this problem, this paper firstly presented the criterions of optimal MapReduce algorithms, which should keep excellent parallelism and optimize all metrics simultaneously. In this paper, we studied sorting algorithm, which is the most important algorithm on data processing, and proposed an optimal sorting algorithm under MapReduce paradigm. Then we illustrated the effectiveness and efficiency by the theory and experiments.

Keywords MapReduce, Optimization, Big data analysis, Sorting algorithm

1 引言

我们处于一个信息爆炸时代,工业界、学术界、政府正在以前所未有的速度积累着大量数据。这使得大数据处理成为最为紧迫的需求,即在TB级甚至更大规模数据上进行快速计算。近年来,为应对大数据分析处理挑战,数据库研究领域正在构建海量并行数据处理平台,一般采用数百甚至更多的普通计算机构成的网络化集群作为硬件处理平台。其中最引人关注的就是MapReduce。

MapReduce[1]自提出以来,不断发展完善,目前已经成为一个相对成熟的计算模式。在高层,一个MapReduce系统由一组无共享机器组成,机器之间通过网络消息传递进行通信。一个MapReduce算法要求这些机器以相互协作的方式共同完成一个计算任务。最初,输入数据集分布在节点之中,典型情况是没有副本的分布方式,即每个数据对象仅在一个节点之上。MapReduce算法以作业为粒度执行(多个作业组成有向无环图),每个MapReduce作业包括3个执行阶段:map、shuflle和reduce。前两个阶段是让机器之间进行数据交换:map阶段负责对数据进行初步处理,准备需要传递到其他节点的输出信息,而shuffle阶段负责实际的数据传输工作。Reduce阶段实际上不需要网络传输,每个节点只需对本地存储的数据进行进一步处理,输出计算结果。到目前为止,当前的作业就已经执行完毕,若计算任务还没有结束,那么需要启动另一后续作业。类似于传统并行计算,一个MapReduce系统的关注点主要包括:负载均衡指标、存储空间开销、每个计算节点的硬件开销(CPU、I/O、网络等)。这些原则在MapReduce算法设计时都曾发挥着重要作用,但之前的研究对这些关注点都只是采取尽力而为的优化原则,很少有多指标同时优化的研究工作。

本文针对以上问题,希望从多角度对算法的性能进行优化研究。针对数据处理领域最重要的排序算法进行理论分析,给出了多指标约束下的最优算法,并证明了该优化算法满足MapReduce优化算法规范。最后通过实验验证了优化的排序算法的有效性和效率。

2 背景知识与相关工作

2.1 MapReduce简介

如前文所述,MapReduce算法以多个作业组成有向无环

到稿日期:2014-01-18 返修日期:2014-03-13
金 菁(1982—),女,硕士生,主要研究方向为软件工程数据挖掘与软件测试等。

图的方式顺序执行，每个作业又分为 map、shuffle 和 reduce 3 个阶段。所有机器都以相同方式执行程序，下面以一节点 M_i 为例来说明这 3 个阶段。

map 阶段。M_i 对本地输入数据进行处理，生成一组输出键-值对(k,v)。键 k 通常是数值类型，而值 v 通常可以包含任意类型的数据信息。在 shuffle 阶段，所有(k,v)键-值对被传递到另外一台节点。

shhuffle 阶段。令 L 表示 map 阶段所有机器输出的键-值对。在 shuffle 阶段，L 分布在一组节点之上，分布遵循的原则为：相同 k 的键-值对必须在同一节点。假设 L 中有$(k_1,v1),(k_2,v_2),(k_3,v_3),\cdots$，它们必须被拷贝到同一节点上处理。

reduce 阶段。reduce 的输入数据是来自于前两个阶段输送的数据，在本地存储系统之中，reduce 阶段就是对这些中间临时数据进一步处理，生成最终的输出结果。该阶段结束后意味着一个 MapReduce 作业结束。

2.2 MapReduce 排序算法定义

设数据集 D 有 P 个数据对象，它们是从有序域中抽取的一组数据对象。当算法结束后，所有的数据对象必须以有序方式分布在 N 个节点上，满足：对任意节点 i、j，若 $i<j$，则 i 上的数据对象应该优先于 j 上的数据对象。

2.3 相关研究

针对 MapReduce 的优化研究工作主要分为两类，即：框架级优化和应用级优化。框架级优化重点关注系统执行框架内部的优化，是针对整体的优化；应用级优化是针对某类具体问题的优化，前提假设是应用运行在同一执行框架之上。因此，框架级优化和应用级优化是相互正交互补的关系。

1)框架级优化研究

Hadoop[3]是目前最流行的 MapReduce 开源实现，它最初是为大规模批处理应用而优化的。Abouzeid 等人针对 Hadoop 不能很好地适应交互式分析的问题，将 Hadoop 与本地数据库结合，提出一个面向大规模交互式分析处理的系统执行框架[4,5]。Dittrich 等人对 Hadoop 进行自定义扩展，用于优化不同类型分析处理操作的执行性能[6]。文献[7,8]通过计算结果重用的方式进一步提升多个 MapReduce 作业的整体执行性能。文献[9,10]讨论了将相关数据保持在同一节点来减少网络 IO 开销。Floratou 等人[11]提出基于列的存储实现并证明了该实现在某些环境下的性能优势。Shinnar 等人[12]建议通过数据尽可能常驻内存的策略来优化磁盘 IO 开销。文献[13,14]设计了一个负载倾斜修正方法，以解决不同节点负载不均衡的问题。

2)应用级优化研究

MapReduce 是一个具有广泛适用性的计算模型，因此，基于它的应用级优化研究也非常广泛。

文献[15]通过对比研究了传统的 join 操作在 MapReduce 环境下的不同算法。Afrati 和 Ullman 提出了一种在 MapReduce 环境下的多路 join 算法优化策略[16]。针对多路 join 算法，Lin 等人[17]采用列存储策略进行优化研究。文献[18,19]研究了集合相似-join 问题。文献[20]针对 theta-join 的处理性能进行深入研究。Bahmani 等人[21]提出了个性化的 page-rank 计算方法。文献[22]研究了最大匹配、点/边覆盖以及最小割问题。文献[23]致力于欧式空间下 k 最近邻-join 问题的研究。文献[24]针对已有的近邻传播聚类算法在面临海量数据时遇到的性能问题，提出了基于 MapReduce 的分布式近邻传播聚类算法——DisAP。文献[25]针对大规模历史数据的高并发数据流处理需求，为改进 MapReduce 的实时处理能力，提出了一种内存 Hash B 树、外存 SSTable 文件的〈key，value〉中间结果缓存技术。文献[26] 针对大规模空间数据的高性能 k-近邻连接查询处理，研究了 MapReduce 框架下基于 R-树索引的 k-近邻连接查询处理，提出了 R-树索引快速构建算法和基于 R-树的并行 k-近邻连接算法。文献[27]基于 MapReduce 模型，提出了一种在线社交网络蠕虫的仿真方法。

3 排序算法分析

3.1 优化算法

令 D 为算法的输入数据集，P 为问题规模，即 D 中数据对象的数量，N 为系统中节点规模。

定义 $d=P/N$ 为每个节点上数据对象的规模，前提假设为 D 在 N 中均匀分布。定义数据集 D 上求解问题的 MapReduce算法 A。当 A 满足以下条件时，称 A 为优化算法。

1)空间约束：对于任意时间点，每个节点使用的存储空间为$O(d)$。这点保证了数据不会在节点之间发生数据倾斜，数据倾斜会造成某些节点负载过重，而某些节点负载相对不足，这将严重影响 MapReduce 的性能[2]。

2)网络约束：对于每一波计算任务，每个节点通过网络发送和接收的数据总量至多为 $O(d)$。该约束条件保证了一个作业在 shuffle 阶段通过网络传输的数据总量至多为 $O(m*N)=O(P)$。数据传输时间近似等于每个节点发送和接收 m 数据规模的时间，因为节点以并行方式传输数据。

3)作业数约束：整体计算任务需要的总作业数应为常数。该约束条件在之前的 MapReduce 算法中也同样如此。它保证了算法在总的网络传输数据总量为 $O(P)$。

4)计算开销：每个节点的总计算开销不超过 $O(\frac{T_{seq}}{N})$，其中 T_{seq} 表示以单节点顺序执行相同计算任务的总开销，即使用 N 个机器并行执行的算法加速比应该为 N。

对于排序问题，输入可以看成具有 P 个数据对象的集合 S，数据均来自于一个有序域。为简化讨论，假设该有序域是实数域，该假设可以很容易地扩展到其他有序域。令 $M_1,\cdots,M_N$ 表示系统中的计算节点，S 分布在它们之中。每个节点存储 $O(m)$规模的数据对象，其中 $m=P/N$。当排序结束时，对于任意 $1\leqslant i<j<N$，所有数据 M_i 中的数据对象必须优先于 M_j 中的数据对象。

3.2 TeraSort

参数 $\rho\in(0,1]$，TeraSort 算法在 MapReduce 模型下可以用如下两个 jobs 来表示。

作业 1

Map-shuffle(ρ)

对每个 $M_i(1\leqslant i\leqslant t)$，以 ρ 为概率对其本地存储的输入数据进行独立采样，将样本数据通过网络发送到 M_1。

Reduce(on M_1)

1. 设 S_{sam} 为 M_1 接收的采样数据集，$s=|S_{sam}|$。

2. 对 S_{sam}进行排序，然后选出一组数据对象 $b_1,\cdots,b_{N-1}$，其中 b_i 是 S_{sam}中第 $i*\lceil s/N\rceil$小的数据对象，$1\leqslant i\leqslant N-1$，每个 b_i 都是一个边界对象。

作业 2

Map-shuffle（假设 $b_1,\cdots,b_{N-1}$被传送到每个计算节点）

每个 M_i 将其本地存储的、在$(b_{j-1},b_j]$之间的数据对象传送到 M_j，其中 $1\leqslant j\leqslant N$，$b_0=-\infty$且 $b_N=\infty$。

Reduce：

每个 M_i 对前一阶段接收到的数据对象进行排序。

为方便起见，上述描述有时会要求某个机器 M 将数据发送到它自己本身。显然，这种数据传输方式发生在 M 节点内部，因此不需要网络数据传输。将上述算法中作业 2 的假设称为广播假设，后文将对其进行详细分析讨论。

在文献[50]中，参数 ρ 的取值是开放式的。然而，ρ 的值将对算法性能产生重要影响。因此，下面将详细分析 ρ 值对算法效率的影响。

3.3 ρ 值选取

定义 $S_i=S\cap(b_{i-1},b_i]$，$i\in[1,N]$。对于作业 2，S_i 中所有的对象都聚集到节点 M_i 上，M_i 通过 reduce 阶段对这些数据对象进行排序。对于 TeraSort，要使算法最优化，必须满足以下两个条件：

C_1：$s=O(m)$

C_2：$|S_i|=O(m)$，$i\in[1,t]$

条件 C_1 需要被满足是因为 M_1 在作业 1 的 map-shuffle 阶段需要接收 $O(s)$规模的数据对象，只有这样才能 3.1 节提出的优化算法的网络约束条件。C_2 需要被满足的原因是在作业 2 处理阶段 M_1 必须要接收并存储 $O(|S_i|)$规模的数据对象，为保证网络约束和数据存储空间约束，该数据规模应为 $O(m)$。

现在给出一个关于 TeraSort 算法的重要事实依据。

定理 1 当 $m\geqslant N\ln(PN)$时，若将 ρ 设置为$\frac{1}{m}\ln(PN)$，则 C_1 和 C_2 同时被满足的概率至少为 $1-O(\frac{1}{P})$。

证明：当 $N<9$ 时，$m=\Omega(P)$，这种情况下，同时满足 C_1 和 C_2 是非常容易的。当 $N\geqslant 9$ 时，证明了切尔诺夫界和分桶理论。

首先，很容易得到式(1)。

$$E[s]=m\rho N=N\ln(PN) \tag{1}$$

由切尔诺夫界理论可得到式(2)，其中最后一个不等式是依据 $N\geqslant 9$ 这个事实。这意味着 C_1 不成立的可能性至多为 $1/P$。

$$\Pr[s\geqslant 1.6*N\ln(PN)]\leqslant\exp(-0.12*N\ln(PN))\leqslant\frac{1}{P} \tag{2}$$

下面在 $s<1.6\ln(PN)=O(m)$情况下分析 C_2 条件的可满足性。

假设 S 按照升序排序，以尽可能均等的方式将其切分为 $\lfloor N/8\rfloor$个子集，每个子集称为一个桶，每个桶中的数据对象介于 $8P/N=8m$ 和 $16m$ 之间。可以发现一个规律：对于每个桶而言，若它至少包含一个边界数据对象 b_i，那么条件 C_2 可以被满足。究其原因，可以发现：在这种情况下，每个桶会掉入两个连续的边界对象之间(包括 b_0 和 b_N)。因此，对于每个 S_i，$i\in[1,N]$，其至多包含两个桶的数据对象，即 $|S_i|\leqslant 32m=O(m)$。

对于一个桶 β，若其至少包含 $1.6\ln(PN)>s/N$ 个样本数据，则其一定会包含一个边界数据对象 b_i，因为边界对象是从$\lceil s/N\rceil$个连续样本对象中取出的一个。令 $|\beta|\geqslant 8m$ 是 β 中数据对象的总规模。

定义 1 若 β 中第 j 个数据对象被选中作为样本，随机变量 x_j($j\in[1,|\beta|]$)的值为 1，否则为 0。则可得式(3)：

$$X=\sum_{j=1}^{|\beta|}x_j=|\beta\cap S_{sam}| \tag{3}$$

显然，$E[X]\geqslant 8m\rho=8\ln(PN)$。可进一步得出式(4)：

$$\begin{aligned}\Pr[X\leqslant 1.6\ln(PN)]&=\Pr[X\leqslant(1-4/5)8\ln(PN)]\\&\leqslant\Pr[X\leqslant(1-4/5)8E[X]]\\&\leqslant\exp(-\frac{16}{25}\frac{E(X)}{3})\quad\text{(切尔诺夫界)}\\&\leqslant\exp(-\frac{16}{25}\frac{8\ln(PN)}{3})\\&\leqslant\exp(-\ln(PN))\\&\leqslant\frac{1}{PN}\end{aligned} \tag{4}$$

上述公式推导证明了如果 β 没有包含边界对象，那么它失效的可能性至多为 $1/PN$。由于最多有 $t/8$ 个桶，每个桶失效的概率至多为 $1/8P$，因此，在 $s<1.6N\ln(PN)$的情况下，条件 C_2 不被满足的概率至多为 $1/8P$，也就是说整体概率最多为 $9/8P$。

那么，C_1 和 C_2 同时得到满足的概率至少应为 $1-17/8P$。

3.4 理论分析

对于较大的 P 值，理论 1 成立的概率为 $1-O(1/P)$。由于 $O(1/P)$接近于 0，可以忽略不计，因此条件 C_1 和 C_2 几乎绝不会不成立。

对于 TeroSort，$m\geqslant N$ 是一个隐含条件，故定理 1 中关于 m 的条件与一个对数函数紧密相关。同时还可以发现：作业 1 的 reduce 阶段和作业 2 的 map-shuffle 阶段都需要用一个节点存储 $N-1$ 个边界数据对象。

事实上，典型情况下有 $m\geqslant N$，也就是说一个节点的内存规模远大于节点规模。更特别地，m 的数量级至少是 10^6(每个节点几 MB 数据规模的情况)，而 P 的数量级是 10^4 甚至更少。因此，它是一个非常合情合理的假设，这也解释了为何 TeraSort 在实践中具有优秀的性能表现。

下面给出最优的 TeraSort 算法，广播假设作为先验条件。条件 C_1 和 C_2 要求每个机器在任意时刻都只能存储 $O(m)$规模数据对象，满足优化算法的空间约束条件。对于网络开销，当算法开始运行时，在每个作业过程中，节点 M 只能发送其自身已经存储的数据对象。因此，在每个作业执行过程中，M 将通过网络传送 $O(m)$规模的数据。进而，由条件 C_1，节点 M_1 仅接收了 $O(m)$规模的数据。因此，优化算法网络约束条件可以被满足。显然，作业数约束条件也可被满足。

4 实验验证

本节给出排序算法的性能对比测试，以验证优化算法的效率。

4.1 集群配置

采用内部私有集群的形式作为算法的验证平台。集群由57个节点构成，一个是主控节点，其他56个是从节点，每个节点采用4个主频为2.4GHz的中央处理器，24GB内存。采用Hadoop1.0版本，为每个java虚拟机分配4GB内存空间，即每个任务最多使用4GB内存。表1列出了Hadoop的重要参数。

表1 Hadoop配置参数

参数名称	参数值
fs. block. size	128MB
dfs. replication	3
io. sort. factor	300
io. sort. record. percentage	0.1
io. sort. spill. percentage	0.9

4.2 数据集描述

采用人工生成的数据集pageview，总规模为330GB，包含约120亿元组，每个元组对应一个Wikipedia页面，记录了该页面在给定时间段被浏览的次数。所有元组按照时间戳排序。对该数据集采用不同的数据分布，用于验证数据分配对算法性能的影响。对于每次实验，采用多次去平均的方法来获得稳定结果。

4.3 实验结果分析

第一组实验比较了3.2节建议的TeraSort算法（简称TS）与Hadoop系统默认的排序算法（表示为HS）。

在给定数据集下，设数据集被分成 k 个数据块分布在Hadoop分布式文件系统之中。对于HS，主控节点首先收集每个数据块中前$\lceil 10^5/k \rceil$个记录，放入集合 S 中，成为样本数据集。然后，主控节点从 S 中挑出 $t_{slave}-1$ 个边界记录（b_1，b_2，…，$b_{slave-1}$），其中 b_i 是 S 中第 $i\lceil 10^5/t_{slave} \rceil$小记录，$t_{slave}$ 是从节点规模数。主控节点启动算法中的第一轮作业（即作业1），将$(b_{i-1}, b_i]$之间的所有记录发送到第 i 个节点进行排序，其中 $i\in[1, t_{slave}]$。显然，HS算法的效率取决于 S 中数据的分布，如果其分布与全数据集相同，那么每个从节点的排序工作量近似相等。否则，将出现负载倾斜，即有些节点需要处理的数据很少，有些很多。

图1给出了两个排序算法执行时间的对比结果。从对比结果可以发现，TS算法严格优于HS算法。随着数据规模的增加，这种性能差距呈线性增加。图2进一步给出了两个算法的最大本地数据对比，可以发现，随着数据规模增加，HS算法的节点级最大本地数据规模急剧增大，而TS算法则增加很缓慢。这种现象说明了TS算法的负载均衡性更好，HS算法将大量数据发送到了少数节点中，造成了负载倾斜。

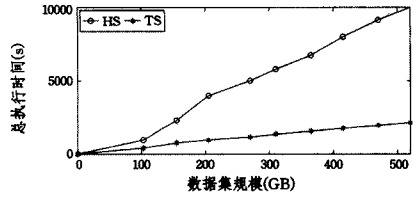

图1 数据分布均衡情况下算法执行时间比较

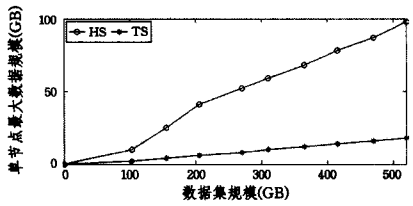

图2 数据分布均衡情况下一个计算节点上最大本地数据规模比较

为进一步验证HS的低效性，图3和图4给出了不同数据分布情况下算法的处理时间和节点级最大本地数据规模。通过对比图1、图2以及图3、图4，可以发现较差的数据分布对TS算法几乎没有影响，而对HS则影响很大。主要原因在于TS的采样过程对原始数据的分布特性不敏感，而HS则非常敏感。

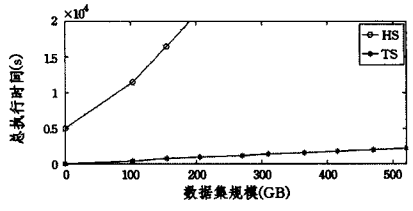

图3 数据分布不均衡情况下算法执行时间比较

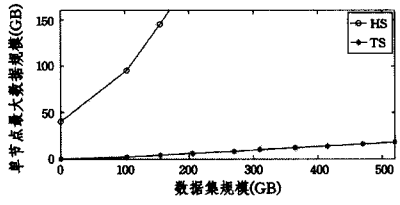

图4 数据分布不均衡情况下一个计算节点上最大本地数据规模比较

最后，验证了样本数据规模对TS算法效率的影响。图5、图6给出了当样本数据规模不断增加时TS算法的执行时间和节点级最大本地数据的变化情况。从图5可以发现，当样本规模为 $N\ln(PN)$时TS算法的执行时间最短，样本规模低于该规模后，将造成严重的数据分布不均衡。图6则以节点级最大本地数据规模的方式说明了 $N\ln(PN)$是样本数据规模的临界值，当样本数据规模大于该值后，数据均衡性都比较好。

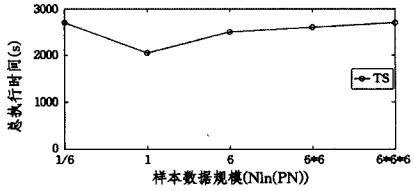

图5 样本数据规模对TS算法执行时间的影响

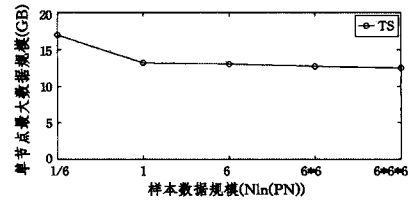

图6 在TS算法下，采样数据规模对一个节点上最大本地数据规模的影响

结束语 MapReduce已经成为大规模并行计算领域非常流行的计算模型。基于MapReduce的算法处于不断的研究发展中，很多算法都没有得到完美的优化。排序在MapReduce数据处理方面是非常经典且重要的操作，很多应用都用到该操作。目前，基于MapReduce模型的排序性能优化理论还不够深入，在实际应用中，很多情况下，算法不能达到最优的执行效果。其主要原因在于，采样和数据分布对并行负载均衡造成了很大影响。传统的排序算法不能灵活智能地解决负载倾斜问题。本文对MapReduce下排序操作的性能优化进行了深入研究。

参考文献

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[2] Kwon Y, Balazinska M, Howe B, et al. Skewtune: mitigating skew in mapreduce applications[C]//SIGMOD. 2012: 25-36

[3] Hodoop Wiki. Apache Hadoop[OL]. http://hadoop.apache.org

[4] Abouzeid A, Bajda-Pawlikowski K, Abadi D, et al. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 922-933

[5] Abouzied A, Bajda-Pawlikowski K, Huang J, et al. HadoopDB in action: building real world applications[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management Df data. ACM, 2010: 1111-1114

[6] Dittrich J, Quiané-Ruiz J A, Jindal A, et al. Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing) [J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 515-529

[7] Elghandour I, Aboulnaga A. ReStore: reusing results of MapReduce jobs[J]. Proceedings of the VLDB Endowment, 2012, 5(6): 586-597

[8] Nykiel T, Potamias M, Mishra C, et al. MRShare: sharing across multiple queries in MapReduce[J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 494-505

[9] Eltabakh M Y, Tian Y, Ozcan F, et al. CoHadoop: flexible data placement and its exploitation in Hadoop[J]. Proceedings of the VLDB Endowment, 2011, 4(9): 575-585

[10] He Y, Lee R, Huai Y, et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems[C]// 2011 IEEE 27th International Conference on Data Engineering (ICDE). IEEE, 2011: 1199-1208

[11] Floratou A, Patel J M, Shekita E J, et al. Column-oriented storage techniques for MapReduce[J]. Proceedings of the VLDB Endowment, 2011, 4(7): 419-429

[12] Shinnar A, Cunningham D, Saraswat V, et al. M3R: increased performance for in-memory Hadoop jobs[J]. Proceedings of the VLDB Endowment, 2012, 5(12): 1736-1747

[13] Gufler B, Augsten N, Reiser A, et al. Load balancing in mapreduce based on scalable cardinality estimates[C]// 2012 IEEE 28th International Conference on Data Engineering (ICDE). IEEE, 2012: 522-533

[14] Kolb L, Thor A, Rahm E. Load balancing for mapreduce-based entity resolution[C]//2012 IEEE 28th International Conference on Data Engineering (ICDE). IEEE, 2012: 618-629

[15] Blanas S, Patel J M, Ercegovac V, et al. A comparison of join algorithms for log processing in mapreduce[C]// Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010: 975-986

[16] Afrati F N, Ullman J D. Optimizing multiway joins in a map reduce environment[J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(9): 1282-1298

[17] Lin Y, Agrawal D, Chen C, et al. Llama: leveraging columnar storage for scalable join processing in the MapReduce framework[C]//Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. ACM, 2011: 961-972

[18] Vernica R, Carey M J, Li C. Efficient parallel set-similarity joins using MapReduce[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. ACM, 2010: 495-506

[19] Metwally A, Faloutsos C. V-smart-join: a scalable mapreduce framework for all-pair similarity joins of multisets and vectors [J]. Proceedings of the VLDB Endowment, 2012, 5(8): 704-715

[20] Zhang X, Chen L, Wang M. Efficient multi-way theta-join processing using MapReduce[J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1184-1195

[21] Bahmani B, Chakrabarti K, Xin D. Fast personalized pagerank on mapreduce[C]//Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. ACM, 2011: 973-984

[22] Lattanzi S, Moseley B, Suri S, et al. Filtering: a method for solving graph problems in mapreduce[C]//Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures. ACM, 2011: 85-94

[23] Lu W, Shen Y, Chen S, et al. Efficient processing of k nearest neighbor joins using mapreduce[J]. Proceedings of the VLDB Endowment, 2012, 5(10): 1016-1027

[24] 鲁伟明，杜晨阳，魏宝刚，等. 基于MapReduce的分布式近邻传播聚类算法[J]. 计算机研究与发展，2012，49(8)：1762-1772

[25] 亓开元，韩燕波，赵卓峰，等. 支持高并发数据流处理的MapReduce中间结果缓存[J]. 计算机研究与发展，2013，50(1)：111-121

[26] 刘义，静宁，陈荦，等. MapReduce框架下基于R-树的k-近邻连接算法[J]. 软件学报，2013，24(8)：1836-1851

[27] 和亮，冯登国，王蕊，等. 基于MapReduce的大规模在线社交网络蠕虫仿真[J]. 软件学报，2013，24(7)：1666-1682

[28] 程兴国，肖南峰. 粗粒度并行遗传算法的MapReduce并行化实现[J]. 重庆理工大学学报：自然科学版，2013，27(10)：66-70