基于索引偏移的 MapReduce 聚类负载均衡策略

周华平 刘光宗 张贝贝
(安徽理工大学计算机科学与工程学院 安徽 淮南 232000)

摘 要 MapReduce 作为一种分布式编程模型,被广泛应用于大规模和高维度数据集的处理中。其采用原始 Hash 函数划分数据,当数据分布不均匀时,常会出现数据倾斜的问题。基于 MapReduce 的聚类算法,需要多次迭代且不清楚各阶段 Reduce 的输入数据分布,因此现有的解决数据倾斜的方法并不适用。为解决数据划分的不均衡问题,提出一种当存在数据倾斜时更改剩余分区索引的策略。该方法在 Map 运行的过程中统计将要分给各 reducer 的数据量,由 JobTrackcr 监控全局的分区信息并根据数据倾斜模型动态修改原分区函数;在接下来的分区过程中,Partitioner 把即将导致倾斜的分区索引到其余负载较轻的 reducer 上,使各节点的负载达到均衡。基于 Zipf 分布数据集和真实数据集,将所提算法与现有的解决数据倾斜的方法进行对比,结果证明,所提策略解决了 MapReduce 聚类中的数据倾斜问题,且在稳定性与执行时间上优于 Hash 和基于采样的动态分区法。

关键词 MapReduce,数据倾斜,负载均衡,分布式聚类,索引偏移
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.05.053

Load Balancing Strategy of MapReduce Clustering Based on Index Shift

ZHOU Hua-ping LIU Guang-zong ZHANG Bei-bei
(College of Computer Science and Engineering, Anhui University of Science and Technology, Huainan, Anhui 232000, China)

Abstract MapReduce has been widely used in large-scale and high-dimension datasets as a kind of distributed programming model. Original Hash partition function in MapReduce often occurs data skew when data distribution is not uniform. In the clustering algorithm based on MapReduce, existing solutions for data skew are not applicable because the input data distribution of Reduce is unclear at each stage of multiple iteration. To solve the imbalance problem of data partitioning, this paper proposed a strategy to change the remaining partition index when data is tilted. In Map phase, the amount of data which will be distributed to each reducer is counted, then the global partition information is monitored and the original partition function is dynamically modified according to the data skew model by JobTrackcr, so the Partitioner can change the index of these partitions which will cause data skew to the other reducer that has less load in the next partitioning process, and eventually balance the load of each node. Finally, this method was compared with existing methods on both synthetic datasets and real datasets. The experimental results show this strategy can solve data skew on MapReduce clustering with better stability and efficiency than Hash method and dynamic partitioning method based on sampling.

Keywords MapReduce, Data skew, Load balance, Distributed clustering, Index shift

1 引言

移动互联网、物联网等信息技术的普及给人们的生活带来了便利,但与此同时,它们每天都会产生 TB 级的数据量,传统的集中式数据处理方式已无法适应如此海量的数据环境。谷歌提出的分布式处理模型 MapReduce 在海量数据的处理过程中表现出了较好的并行性和扩展性[1],它将整个分布式计算过程分为 Map 和 Reduce 两个阶段,隐藏了节点间的信息交流与数据传输,用户只需要完成对 Map 和 Reduce 函数的设计与实现就可以在集群中处理数据。Apach 的开源项目 Hadoop 将 MapReduce 建立在具有较强容错能力的分布式文件系统 HDFS 上,在互联网的大数据处理企业中得到广

到稿日期:2017-03-06 返修日期:2017-06-04 本文受国家自然科学基金(51174257),安徽理工大学矿业企业安全管理研究中心招标项目(SK2015A084),安徽省高校优秀青年人才支持计划项目资助。

周华平(1979-),女,博士,教授,主要研究方向为大数据、数据挖掘,E-mail:hpzhou@aust.edu.com(通信作者);**刘光宗**(1995-),男,硕士生,主要研究方向为大数据、数据挖掘,E-mail:819461419@qq.com;**张贝贝**(1988-),女,硕士生,主要研究方向为大数据、数据挖掘,E-mail:976982000@qq.com。

泛应用并迅速发展成分析大数据的领先平台[2]。

在 MapReduce 的并行化处理中,任务的完成时间取决于最慢的 Reduce 的运行时间,数据倾斜是影响其性能的重要因素。产生数据倾斜的原因可以分为两种:1)节点获得的数据量不均衡;2)节点对中间结果的的的处理代价不同[3]。中间结果的处理代价在连接计算中影响较大,本文策略主要考虑 MapReduce 聚类算法中各节点数据量的均衡。当发生数据倾斜时,分得数据量较多的节点的执行时间要长于平均时间,而其他节点必须保持空闲以等待倾斜节点处理完成,这使得整个任务的执行时间延长,降低了集群效率。文献[4]通过在 Hadoop 平台上进行大量实验发现,使用默认的 Hash 方法,92%的任务出现了 Reduce 端的数据倾斜,而这些 reducer 的运行时间比平均时间多 22%~38%。因此,如何制订分区策略已成为近年来的研究热点。

目前,MapReduce 中解决数据倾斜的方法主要是先采样获取数据集的大致分布,再根据采样结果决定如何将各键值对分配给 Reduce 以达到负载均衡[5]。但是,由于基于MapReduce的聚类算法需要多轮迭代运算,且每轮运算的数据分布各不相同,抽样统计和分布模型的建立也须进行多次[15],这不仅延长了该作业的执行时间,还会消耗大量的系统资源,降低了整个集群任务的并发执行速度。

针对上述问题,本文提出了一种基于索引偏移的分配方法。该方法计算了 Map 过程中发送给各 reducer 的数据量,并将其汇总后交由 JobTracker 管理和监测。当某个 reducer 要接收的数据量超过数据总量的一定比例时,将接下来要发送给该 Reduce 的数据分区索引偏移到其余 Reduce 中,以最终达到各节点数据的均匀分布,实现负载平衡。本文的主要贡献如下:

1)提出了一种索引偏移分配方法,该方法的优点是无需在作业运行前确定数据的分布情况,而是在 Map 阶段根据已分区信息动态地调整分区。

2)在 MapReduce 原有的框架上增加了判断数据倾斜的评价模型,在 Map 阶段任务进行的同时统计发送到各 reducer 的数据量,无须增加一轮采样。

3)建立聚类中心处理模型,每轮迭代中产生的分散聚类中心不在该轮 Reduce 阶段合并,而是在下一轮迭代的 Map 阶段中完成。

本文第 2 节主要介绍基于 MapReduce 的聚类算法与 MapReduce 中数据倾斜的相关工作;第 3 节首先介绍MapReduce 实现聚类算法的具体流程,然后阐述如何通过索引偏移策略来解决其中的数据倾斜问题;第 4 节给出索引偏移分配分区的实现算法与代价模型;第 5 节在真实数据集上将本文算法与现有的 MapReduce 算法进行对比,验证了本文方法的稳定性及有效性;最后对本文策略做出总结并概述进一步的研究方向。

2 相关工作

Dean 等人[1]提出采用 Hash 法对数据进行分区,事实证明该方法不仅分区速度快,而且具有通用性,因此 Hadoop 将 Hash 分区作为其默认分区方法,同时提供用户自定义分区方法。但 Hash 划分在数据量分布不均匀时无法保证各分区数据的均衡性,且用户在没有明确数据的分布时也无法通过重新定义分区函数来消除数据倾斜的影响。为解决该问题,研究者们提出了多种均衡负载的分区方法。现有的分区方法可大致分为两类:1)通过预处理来避免数据倾斜;2)在出现数据倾斜时对数据分片进行重新分区。

文献[6]提出了一种先采样设置均衡分区函数,然后再根据分布完成用户分区的方法。该方法通过抽样法获取数据的大致分布,并根据抽样结果均衡数据分区。Gufler 等人[7-8]提出了一种基于采样的分区划分方法,该方法在 Map 函数中增加了采样函数,减少了系统通信等资源的消耗,当 Map 阶段完成一定比例后,根据采样结果对分区进行分割和合并来平衡数据分配。周家帅[9]在 mapper 运行时首先将中间数据保存起来,通过对该结果进行采样分析来确定 reducer 的个数以及数据均匀分配的方式。但是,该方法在 Map 阶段执行完才开始传输数据,相比 Hadoop 原有的边处理边传输的方式,增加了数据传输的时间。文献[10]提出了一种增量式分区策略,在 Map 阶段每次溢写时减小分区粒度,产生出 reducer 个整数倍的微数据分区,再根据贪心算法将微分区均匀划分到各节点。但是,该方法无法较好地解决单一键值较多导致的倾斜问题。本文方法在 Map 运行过程中动态调节分区函数,减少了采样操作,同时可以较好地解决单一键值问题。

另外,李航晨等人[11]利用 MapReduce 中的混洗阶段计算每个 reducer 的压力值,系统根据数据分布情况重新调度负载较重的节点来分区,以平衡整个集群负载,而无需用户提供额外的输入。Kwon 等人[12]提出了一种在 reducer 运行过程中平衡各节点时处理数据量的方法。该方法为所有 reducer 建立剩余数据的处理代价模型,将一定比例的未完成 reducer 的剩余数据传输到已完成的 reducer 上,从而使倾斜的节点将数据或任务平均到其他节点中,实现整个 MapReduce 任务的数据均衡。以上两种负载均衡方法都是在 Reduce 端获取分区之后将负载较大的节点数据重传到其余节点中,但在 reducer 运行过程中增加了额外的数据传输代价。而本文方法属于倾斜发生前的预处理,不会增加数据传输代价。

自分布式处理模型 MapReduce 被提出以来,学者们就开始研究将其应用于大规模和高维度数据集的聚类分析等处理中,结果发现该模型在接收和处理数据时出现了较严重的数据倾斜问题[13]。文献[14]针对周期性的任务提出了一种基于历史数据来指导 Reduce 划分的方法,但聚类过程中每轮数据的分布与前一轮的分布关系不大,该轮分布无法作为下一轮的划分指标。文献[15]提出先采样获取聚类结果的大致分布,然后根据分布情况分割并组合数据聚集成的类,并将其均匀分到各 reducer 中。但是该方法在聚类的每轮迭代操作前都需要重新进行采样并建立分布模型,增加了多轮采样过程。现有文献提出的 MapReduce 聚类数据均衡方法大多是通过

抽样法来均衡数据分布，增加了多轮采样以及分割合并操作；本文方法则是在 Map 阶段动态调整索引，平衡数据分区，仅需在最后一轮添加合并操作。

3　MapReduce 聚类算法中的数据倾斜问题

本节首先介绍在 MapReduce 中实现聚类算法的一般流程，并解释其中产生的数据倾斜问题，然后阐述通过索引偏移来实现负载均衡的具体流程。

3.1　MapReduce 聚类中的数据倾斜

分布式计算模型 MapReduce 的运行过程可分为 Map 和 Reduce 两个阶段。

Map 阶段：Hadoop 系统根据用户设置的分块大小对输入文件进行分割，以一对一或多对一的方式启动相应个数的 mapper 以获得数据分片；各 mapper 按照定义的 Map 逻辑将数据转换成键值对〈key_1，$value_1$〉，并映射为另一组键值对〈key_2，$value_2$〉；再对 key 值按照系统默认的 Hash 分区函数进行计算，将具有相同分区值的元组传给同一个 reducer 进行相关计算。

Reduce 阶段：各 reducer 通过 Http 方式从 mapper 中获得自身对应的分区，然后对得到的数据分区执行 Reduce() 函数，最终将运算结果写到各自的输出文件中。

并非所有的传统聚类方法都适用于分布式扩展。适用于分布式扩展的聚类算法应具有可用个别数据代表聚类中的一类数据，且不会对聚类结果造成影响的特点。如对于基于密度的聚类方法，由于其侧重于相邻聚类点之间的关系，因此用某一类中的少数数据无法代表该类，经多个节点处理后的结果无法正确、有效地合并，聚类的准确度大大降低；但是若不用少数数据表示各节点的聚类结果，最终的合并操作则会消耗大量的计算时间。在现有的工作中，分布式聚类算法多是针对 K-means 的扩展，接下来以在 MapReduce 实现的 K-means 算法为例来说明其中的数据倾斜问题。

K-means 算法的主要思想是：首先确定 K 个聚类中心，然后根据数据到每个聚类中心的距离把数据分到距离最近的聚类中，等到所有数据被分配完后，重新计算聚类中心并重新划分聚类，直到其聚类中心的变化量不超过设定阈值为止。在分布式计算模型 MapReduce 上实现的聚类算法与传统聚类算法的基本思想类似，在 MapReduce 上实现的一般流程见图 1，具体过程如下。

1）先确定 K 个初始聚类中心$\{C_0, C_2, \cdots, C_{K-1}\}$，并将它们加入分布式缓存；

2）Map 阶段：mapper 把数据分片中的每一条数据划分给距离其最近的聚类中心，形成 K 个类，每条数据所属的类作为该数据的 key；

3）Shuffle 阶段：把每个 mapper 产生的〈key，$value$〉按照默认的分区方式 Hash(key) mod R 分到 R 个 reducer 中，属于同一类的数据分区将被分配到同一个 reducer 中；

4）Reduce 阶段：对于任意的 reducer，把来自不同 mapper 的数据按其所属类进行合并，并计算出新的聚类中心。

将新的 K 个聚类中心值传送给分布式缓存，重复执行步骤 2）－步骤 4），直到聚类中心值的变化量不超过限定阈值。

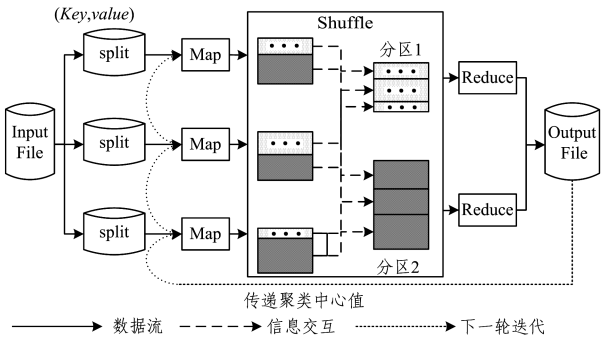

图 1　基于 MapReduce 的 K-means 聚类算法的处理流程

Fig. 1　Processing flow of K-means clustering algorithm based on MapReduce

可以看出，基于 MapReduce 的 K-means 聚类算法是通过执行多轮迭代的 MapReduce 任务来完成的。每一轮任务的 Map 端处理的数据都是针对整个原始数据集的，处理结果为以每条记录所属的类为键、记录中的内容作为值的键值对。而在 Shuffle 阶段，利用 Hash 函数虽然保证了每个 reducer 能分得相同数目的类，但是无法保证每个类包含的数据量都均匀，因此在每轮 MapReduce 任务中都可能存在 Reduce 端的数据倾斜问题。在 MapReduce 模型中，整体任务的完成时间取决于最慢节点的时间，Reduce 端的数据倾斜将导致个别 reducer 的执行时间过长，因此增加了整体的执行时间。而聚类的每轮迭代在获取聚类中心时，各类结果的分布情况都不相同，因此在每次迭代中都须重新平衡倾斜数据。

3.2　基于索引偏移法消除 MapReduce 聚类中数据倾斜的具体流程

MapReduce 现有的分区方法是为了保证键值能均匀分布，因此将包含相同键值的数据分到同一组中，而没有考虑到键包含不同数据量时对整体分区的影响。本文提出的索引偏移方法在原有的 MapReduce 上进行了两处修改：1）在 Map 中增加了各分区的数据计数器，以判断是否产生倾斜分区；2）在提供的 partitioner 接口中加入索引偏移值，使得分区函数可根据具体运行情况进行动态调整。

本文策略在 Map 作业运行中掌握全局分区状态，并根据数据的倾斜状况平衡分区，因此在全局任务的主控部分即 Job Tracker 中增加 3 个模块：全局分区（Global Partition）、倾斜模型（Skew Model）和索引偏移（Index Shift）。其中，全局分区模块用于统计全局作业中 mapper 已经完成的分区情况；倾斜模型用于判断分区是否存在数据倾斜，主要受 Reduce 端需处理的总记录量和所有 reducer 的平均处理速率的影响；索引偏移模块则是在将产生数据倾斜时选择出最佳分配 reducer，计算它与倾斜 reducer 之间的索引偏移量，并将偏移量通过心跳机制发送给所有 Map。这 3 个模块将会在第 4 节中详细介绍。

图 2 为利用索引偏移策略解决 MapReduce 聚类中数据倾斜问题的一轮作业流程。

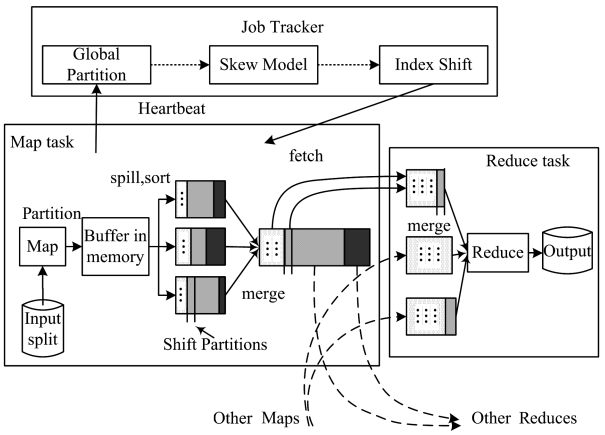

图 2 基于索引偏移法的 MapReduce 聚类的数据均衡

Fig. 2 Data balancing in MapReduce clustering based on index migration

在该轮 MapReduce 作业中，每个 mapper 增加 R 个分区计数器，用于记录 mapper 的结果分区(Partition)分到每个 reducer 的记录量。各 mapper 分区计数器的数值将通过心跳(Heartbeat)机制发送给 Job Tracker 汇总，通过全局信息和倾斜模型来判断出倾斜的 reducer 分区，再从所有非倾斜的 reducer 中选出最小负载 Reduce 接收偏移分区(Shift Partition)，并计算倾斜 reducer 与偏移分区之间的索引差值。Job Tracker 通过心跳将偏移量发送给 Partitioner 以动态修改分区函数，接着将所有原本要发送到倾斜 reducer 的数据加上一个偏移量后划分到其余负载较轻的 Reduce 中。一轮偏移之后，继续统计分区并判断和处理倾斜数据，直到所有数据的 Map 阶段完成，本作业中所有的 reducer 均达到最佳负载。最后，mapper 输出的〈key，$value$〉键值对将被写入内存缓冲区，当缓冲区的数据量超过一定阈值时将发生数据溢写(Spill)并按 key 排序(Sort)，每个 mapper 将所有溢写文件按分区合并(Merge)后写到磁盘上，同时按照不同的划分将结果发送给对应的 reducer。Reduce 端则会合并各个 mapper 输出中拥有同一个分区值的输出，并对合并的结果进行排序，然后交给 reducer 处理。

本文分区方法无需获取各 key 值在输入数据中所占的比例，但是在平衡负载的过程中由于原分区发生了偏移，相同 key 值的 Map 处理结果被发送到不同的 reducer 中，因此应在 Reduce 任务执行完之后对结果进行一次合并操作。而在MapReduce 聚类算法中，Reduce 输出的结果将直接在下一轮的 Map task 初始化时由 Job Tracker 合并后发送到所有 Map，所以只需要在最后一轮运算之后增加合并操作。

4 索引偏移策略的的代价模型与具体算法

本节首先详细介绍实现索引偏移策略的 3 个重要模型，然后给出该算法的具体实现和对算法的代价评估。

4.1 索引偏移策略的代价模型

本文索引偏移策略的主要思想是通过获取所有 mapper 的分区信息来动态更改 partition 函数，修改的 partition 函数是在原 Hash 分区函数的基础上加上对应 reducer 的索引偏移量，使得各 reducer 分得的数据量大致相等。其中，索引偏移量的值由 Job Tracker 中增加的 3 个模块共同求得，分别是全局分区、倾斜模型和索引偏移。首先，获得全局分区情况，即总体的数据量和已处理的数据分区；然后，根据倾斜模型判断倾斜分区；最后，选出最佳偏移 reducer，将其与倾斜 reducer 的索引差值发送给 Partitioner。

原分区函数为 $f(key)=\mathrm{Hash}(key) \bmod R$，$R$ 为 Reduce 的数量。更改后的 partition 函数为 $f'(key)=f(key)+Pf(key)$，其中 $Pf(key)$为倾斜 reducer 随后接收分区的偏移量，所有偏移量的初始值为 0，且随着倾斜分区的出现而变化，$f(key)=\{0,1,\cdots,R-1\}$。

4.1.1 获取全局分区

全局分区通过在 Map()函数中增加计数方法 Count()来获取的M 个 mapper 的分区信息汇总，Count 方法中包含一个计数器数组$\{count_0,count_1,\cdots,count_{R-1}\}$来统计 R 个 reducer 将获得的分区中记录的数量。在该动态分区方法中，应针对 mapper 的输出即〈key，$value$〉中的 key 值做带有偏移量的 Hash 运算，并将结果所属的 reducer 分区对应的计数器加 1，即：

$$count_i=\sum_{f'(key)=i}(key,value) \tag{1}$$

其中，$i=\{0,1,\cdots,R-1\}$，则每个 reducer 分得记录的总数量 CR_i 为：

$$CR_i=\sum_{i=0}^{M-1}count_i \tag{2}$$

其中，M 为 mapper 的个数。所有 mapper 通过心跳机制将计数器信息发送给 Job Tracker，汇总得到每个 reducer 获得的记录数，这些全局记录 CR_i 是判断数据倾斜的基本条件。

4.1.2 数据倾斜模型

本文的数据倾斜模型建立在全局分区的基础之上，考虑了因硬件差异等因素导致的 reducer 处理速率的差异。reducer 的处理速率为 CV_i，其代表各 reducer 在单位时间内处理的记录数，在 Map 任务启动时，由各 reducer 计算完成后发送到 Job Tracker。在 Map 阶段完成之前，若某个 reducer R_i 已获得的分区中的记录数 CR_i 与其处理速率 CV_i 的比值超过了所有 reducer 需处理的总记录数 S_C 与平均处理速率的比值，则将其视为倾斜 reducer，即：

$$\frac{CR_i}{CV_i}\geqslant\frac{R*S_C}{\sum_{i=0}^{R-1}CV_i},R_i\notin R_S \tag{3}$$

此时应将该 reducer(即 R_i)加入存储倾斜 reducer 的集合 R_S 中，集合 R_S 中的 reducer 在下次倾斜判断中无须考虑，R_S 初始为空。

但是在 Map 作业处理结束前，用户只能获得输入的总数据量 S_D，而原 MapReduce 框架的内置计数器包含对所有 mapper 已处理的字节数 O_D 和已输出记录数 O_C 的统计，内置计数器则由 Job Tracker 来维护。作业运行中，多个 mapper 同时处理输入数据集的不同分片，已处理的数据可视为随机抽样处理，即整体的记录数与已处理数据记录存在一定的比例关系，因此 reducer 需处理的总数据记录数为：

$$S_C=\frac{S_D*O_C}{O_D} \tag{4}$$

每处理一定比例的数据则重新计算一次总记录数 S_C'：

$$S_C = \max\{S_C, S_C'\} \tag{5}$$

在 Map 任务的处理中，随着已处理数据的增加，整体记录量的估计值趋向于实际值。在该数据倾斜模型中，只有当已处理数据量超过总数据量的 $1/R$ 时才开始进行倾斜判断，这是因为在该界限之前所有 reducer 都不会发生倾斜，且当处理数据过少时，根据比例关系求解会产生较大误差。

4.1.3 索引偏移平衡分区

当原分区存在数据倾斜问题时，不应再为该倾斜 reducer 分配分区。在一轮索引偏移平衡分区中，需要进行以下处理。

1）求出本轮倾斜 reducer 的偏移值。如果第 i 个 reducer（即 R_i）为倾斜 reducer，那么应从倾斜集合 R_S 的补集中选择出负载最小的 reducer 接收 R_i 的倾斜分区，它们之间的差值即为 R_i 的偏移量 P_i。其中，P_i 为偏移量数组 $\{P_0, P_1, \cdots, P_{R-1}\}$，表示 R 个 reducer 在倾斜时分区索引应该偏移的量，偏移量均初始为 0。已获得的分区中记录数 CR_i 与其处理速率 CV_i 的比值最小的 reducer 即为最小负载 reducer。

2）修改已有偏移值。每个 reducer 都有一个偏移集合 R_T 来指示倾斜分区的偏移过程，当被选中接收倾斜分区的偏移 Reduce 被添加到倾斜 R_i 的偏移集合时，偏移集合中包含 R_i 的 reducer，且其原偏移值要加上 R_i 的偏移值 P_i，即原来的索引偏移到该倾斜 reducer 中的分区须产生新的偏移。该集合至少包含一个 reducer，即其本身。整个作业运行结束时，各偏移集合至多会包含 R 个 reducer，至少会含有一个自身元素。

3）将偏移值发送到分区函数。将偏移量数组通过 Heartbeat 发送给整个任务的 Partitioner，供其动态调整分区以平衡负载。

综上所述，索引偏移策略发生在 Map 的运行阶段。每一轮选择 reducer 接收偏移分区就意味着有一个 reducer 的数据量达到本次 MapReduce 作业总处理量的平均值，因此至多需要执行 $r-1$ 次偏移数组的修改就可完成所有的数据分配过程。

4.2 索引偏移策略的算法实现与分析

本文提出的算法首先在 Map 任务执行期间对 mapper 中分到各 reducer 的记录数进行统计，并将其发送到 Job Tracker 中；然后，Job Tracker 根据 reducer 的个数、记录数以及数据量等信息判断出即将产生倾斜的 reducer，并从非倾斜 reducer 中选择负载最小的 reducer 作为偏移 reducer 来接收倾斜分区，它们之间的差值则是分区的索引偏移量；最后，将偏移量发送到 Partitioner，以动态修改分区函数。循环上述步骤直至 Map 任务运行结束，Reduce 端将不再出现数据倾斜问题。索引偏移策略详见算法 1。

算法 1　偏移索引动态分区

输入：所有 mapper 统计的各分区数据记录量

输出：各 reducer 分区的偏移量

```
1. WHILE((OutData/SumData >= 1/R) && (!Map. Complete))
   /* Map 已处理数据超过总数据的 1/R 且未完成 */
2. FOR i=0 TO R-1 /* R 个 reducer */
3.   IF !ReduceSkewSet. Contain{Reducei};/* 已处理的倾斜 reducer 不考虑 */
4.   CountRecordi = Sum{ Reducei. AllValue };/* 第 i 个 reducer 已得到的总数据记录 */
5.   SumCount = max{SumCount, SumData * OutCount/OutData};
     /* 本次作业的总记录数 */
6.   IF CountRecordi/CVi >= SumCount * R/Sum{CVi} /* reducer 端出现倾斜 */
7.     ReduceSkew. Add{Reducei};/* Ri 加入倾斜 reducer 集合 */
8.     MinLoad=Min{Reduce. Except{ReduceSkewSet}};/* 非倾斜 reducer 中负载最小的节点作为偏移 reducer */
9.     SkewShift =Reducei. Index-MinLoad. Index;/* 倾斜 reducer 的偏移量 */
10.    FOR k=0 TO R-1
11.    IF Reducek. ShiftSet. Contain{Reducei} /* 偏移集合中包含 Ri 的倾斜 reducer */
12.      Shiftk+=SkewShift ;/* 改变指向倾斜 reducer 的偏移索引值 */
13.      Reducek. ShiftSet. Add{ MinLoad };/* 偏移 reducer 加入倾斜 reducer 的偏移集合中 */
14.    ENDIF
15.    ENDFOR
16.  ENDIF
17.  ENDIF
18. ENDFOR
19. RETURN Shift;/* 输出索引偏移值数组 */
20. ENDWHILE
```

该算法中第 2－18 行所示的循环是索引偏移策略的核心部分，从第 3 行可以看出，循环中每次操作不考虑已处理过的倾斜 reducer，因此对倾斜的处理最多只会执行 $R-1$ 次。而第 4－6 行则通过全局的 Reduce 分区信息、总数据量和已处理数据记录等判断出倾斜的 reducer，其中全局分区是将 M 个 mapper 的统计信息进行求和；第 7－9 行计算倾斜 reducer 到最小负载 reducer 的偏移量，遍历所有 reducer 即可找出最小负载 reducer；第 10－15 行的循环对已有倾斜 reducer 的索引偏移值进行更新，最多更新 $R-1$ 个 reducer 的索引。因此，本算法的时间复杂度为 $O(MR)$，受 mapper 和 reducer 个数的影响，一般情况下 $M>R$。本文策略不用为每个 key 维护一个统计信息，只需为每个 reducer 保存一个计数器即可，大大减少了内存等资源的消耗。

5 实验结果与分析

本节在真实数据集下针对 Hash 法、采样法以及本文方法进行了对比实验，结果表明本文的索引偏移策略可以稳定实现 MapReduce 聚类中各结点的数据均衡。

5.1 实验设置

1）集群环境。整个 Hadoop 集群由 4 个节点构成，包括 1 个 Master 节点和 3 个 Slave 节点，节点之间通过百兆网络连接，且每个节点的配置相同：硬盘 500 GB，内存 4 GB，处理器为双核 Inter(R) Pentium(R) CPU @3.4 GHz，操作系统为 Ubuntu13.0，安装 Hadoop 版本 2.0.0，采用 Java JDK1.7 编译。

2）实验数据。本文实验数据集包括两类：真实数据集和合成数据集。合成数据集为 200 M 满足标准 Zipf 分布的[1，1000]之间的正整数，按元素个数降序排列，第 k 位元素所占比例为 $k^{-\alpha}/\sum_{i=1}^{N} i^{-\alpha}$，其中 N 为总元素数，α 表示数据的倾斜程

度，α 越大则数据越倾斜，当 $\alpha=0$ 时数据集为标准均匀分布。真实数据集为从数据共享平台——数据堂获取的 10010 篇包含事件的新浪微博，包括社会类 2527 篇，娱乐类 1879 篇，科技类 1135 篇，法制类 903 篇，教育类 369 篇，文化类 505 篇，经济类 594 篇，体育类 709 篇，政治类 228 篇，环境类 244 篇，其他类 917 篇。

3)实验方案。在本文实验中，通过在标准 Zipf 合成数据集上的对比实验来展示本文方法的负载平衡性能，以经典的 WordCount 为例，测试 α 值在 0 到 3 之间以 0.5 为增量的程序运行时间；对于真实数据集，本实验采用了现有的基于 MapReduce 优化的 K-means++聚类算法，对比了 Hadoop 系统默认方法、基于采样的动态分区法及本文的索引偏移法的 MapReduce 聚类运行时间。

5.2 实验结果

5.2.1 负载平衡性能

本实验通过 WordCount 程序在标准 Zipf 数据集上对比了 Hadoop 默认的 Hash 方法、基于采样的动态分区法 DPBS[9] 和本文方法 IS(Index Shift)的数据均衡能力。其中，Hash 分区方法将得到相同 Hash 结果的 *key* 分配到相同 reducer；DPBS 首先对 Map 结果采样并按 *key* 进行统计，之后使用贪心算法平衡采样数据得到动态划分函数，最后 reducer 根据该函数获得数据分区；本文方法 IS 则在作业启动时更新倾斜模型，然后在 Map 阶段根据倾斜模型和已处理记录的分区信息动态修改分区函数，并将数据均衡分配到各 reducer。

图 3 比较了这 3 种方法在不同倾斜度情况下的运行时间，该实验中 reducer 的数量被设置为 6，图中横轴表示数据集的倾斜度 α，纵轴表示作业的运行时间。当数据分布完全均匀，即$\alpha=0$时，Hash 方法处理的时间最短，这是由于其余两种方法增加了对数据的倾斜处理；当数据分布较为均匀，即倾斜度$\alpha<0.5$时，DPBS 方法和 IS 方法的执行时间略长于 Hash 方法，但是 3 种方法的执行时间相差不大，是因为 DPBS 方法在 Map 结束时增加了采样处理和数据传输，IS 方法则是在 Reduce 阶段之后增加了合并操作；当数据分布不均匀度增加，即 $\alpha>0.5$ 时，随着 α 的增大，DPBS 方法和 IS 方法的执行时间相对于 Hash 方法显著减少。对于 Hash 方法，随着数据倾斜程度的增加，其总体运行时间增长得较快，即受数据倾斜的影响较大；而采用 DPBS 和 IS 两种方法的作业受数据倾斜的影响较小，且对于 α 的各阶段的取值，IS 方法的处理时间均短于 DPBS 方法，这是因为 IS 方法少了一轮抽样统计操作。综上可以看出，对于不同倾斜度的数据集，IS 方法都能展示出较好的平衡效果。

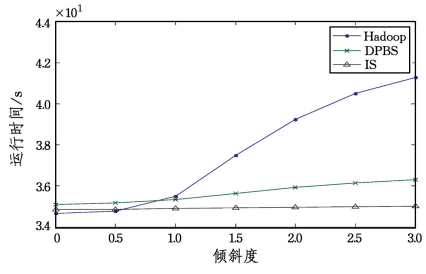

图 3 各算法在标准 Zipf 数据集上的运行时间的对比

Fig. 3 Contrast of running time of algorithms on standard Zipf data sets

5.2.2 MapReduce 聚类中运行时间的优化效果

本实验采用基于 MapReduce 优化的 K-means++聚类算法，分别在 Hadoop 原有的 MapReduce 以及 DPBS 和 IS 改进的 MapReduce 模型中对 10010 篇新浪微博进行分析聚类。为保证变量的单一性，将所有实验的 K 值固定；同时，选取相同的初始聚类中心，以保证聚类轮数以及每轮聚类的分布相同。为在分布式集群中获得较好的实验效果，将真实数据集预处理之后放大 100 倍。图 4 比较了 3 种方法在基于 MapReduce 的聚类迭代中各阶段程序执行的时间，同时取 10 次实验的平均值作为最终的结果，将该实验的 reducer 数量设置为 6，图中横轴表示聚类迭代轮数，纵轴表示作业的运行时间。对于 Hash 方法聚类，每轮的执行时间相对于其余两种方法波动较大，这是因为每轮的聚类分布不同将导致各结点的负载倾斜程度发生变化；DPBS 方法的平均执行时间比 Hash 方法的短，且每轮执行时间较为接近，无较大波动；IS 方法每轮的 MapReduce 任务执行时间均比 DPBS 短，这是因为在平衡数据的过程中节省了抽样统计操作，且只在最后一轮任务结束后增加一次结果合并，执行效率高于 DPBS 方法。

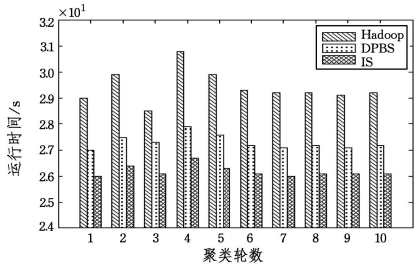

图 4 聚类算法各轮 MapReduce 作业运行时间

Fig. 4 Running time on MapReduce for each round of clustering algorithms

从图 4 可以看出，从第 6 轮开始，3 种方法每轮的 MapReduce任务执行的时间均趋于平稳，因此选择前 5 轮 MapReduce任务的详细数据来进一步分析 Reduce 端的负载情况，结果如表 1 所列。

表 1 每轮聚类中最大负载的 reducer 处理的记录量及运行时间

Table 1 Record amount and running time of maximum load reducer for each round

聚类轮数	Hash		DPBS		IS	
	记录数	运行时间/s	记录数	运行时间/s	记录数	运行时间/s
1	178600	8.2	136900	6.2	135800	6.2
2	142700	6.5	140300	6.4	137700	6.2
3	190200	7.6	138200	6.2	139500	6.3
4	243400	9.8	138500	6.3	138700	6.2
5	231800	9.2	136100	6.2	137400	6.2

从表 1 可以看出，每轮聚类中最大负载的结点的负载情况都不相同，因此每轮的 MapReduce 聚类任务都应单独考虑。在 Hash 方法中，reducer 的执行时间最大相差近 1.6 倍，而经过 DPBS 方法和 IS 方法平衡后，该比例分别为 1.04 和 1.03。IS 方法与 DPBS 方法的最大负载 reducer 处理的数据量与执行时间都较为接近，说明本文方法的动态平衡负载能力与 DPBS 方法的采样预处理效果相差不大。若仅考虑 Re-

duce 任务，相对于 Hash 方法，IS 方法的平均执行时间缩短了 25%。

图 5 主要展示在总处理资源不变的情况下，用户设置的 reducer 个数对整体 MapReduce 聚类执行时间的影响。随着 reducer 个数的变化，Hash 方法的处理时间出现较大波动，尤其在某些 reducer 个数设置不合理的情况下延长了总体执行时间，这是由于最大负载节点产生了较为严重的数据倾斜，使得其执行效率降低；对于不同的 reducer 个数的设置，DPBS 方法的平均运行时间较 Hash 方法短，但在个别情况下其运行时间却长于 Hash 分区法，这是由于其增加了采样的消耗，在 reducer 数量设置得较为合理时其执行效率低于 Hash 方法。可以看出，IS 方法的整体执行时间受用户定义的 reducer 个数的影响较小，比 DPBS 方法短，且更加接近 Hash 分区较为均匀的情况，表现出了较好的负载平衡效果。

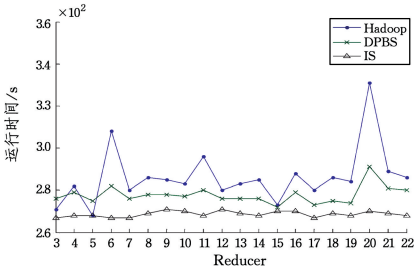

图 5　Reducer 个数对聚类时间的影响

Fig. 5　Effect of reducer number on clustering time

结束语　本文提出了一种索引偏移的策略来解决基于 MapReduce 框架的聚类算法中的数据倾斜问题。该方法不需要事先了解数据的分布情况，而是在 Map 阶段根据已分区数据由 JobTracker 动态地调整分区，从而实现 Reduce 段的负载均衡。最后，在标准 Zipf 数据集和真实数据集上验证了该方法的有效性和高效性。本文主要考虑数据量的均衡分布，未考虑计算量的差异，下一步将对 MapReduce 负载均衡做更全面的研究。

参 考 文 献

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

[2] WANG S, WANG H J, QIN X P, et al. Architecting big data: challenges, studies and forecasts[J]. Chinese Journal of Computers, 2011, 34(10): 1741-1752. (in Chinese)
王珊，王会举，覃雄派，等. 架构大数据：挑战、现状与展望[J]. 计算机学报，2011，34(10)：1741-1752.

[3] ZHANG C C. Design and Optimize Big-Data Join Algorithms using MapReduce[D]. Anhui: University of Science and Technology of China, 2014. (in Chinese)
张常淳. 基于 MapReduce 的大数据连接算法的设计与优化[D]. 安徽：中国科学技术大学，2014.

[4] LIN J. The curse of Zipf and limits to parallelization: A look at the stragglers problem in MapReduce[C]// 7th Workshop on Large-Scale Distributed Systems for Information Retrieval. Boston, USA, 2009.

[5] GENG Y J. The Research of Skew with Counter Technique in MapReduce[D]. Dalian: Dalian Maritime University, 2013. (in Chinese)
耿玉娇. MapReduce 中基于抽样技术的倾斜问题研究[D]. 大连：大连海事大学，2013.

[6] RACHA S C. Load Balancing Map-Reduce Communications for Efficient Executions of Applications in a Cloud[D]. Bangalore: Indian Institute of Science, 2012.

[7] GUFLER B, AUGSTEN N, REISER A, Kemper A. Load balancing in MapReduce based on scalable cardinality estimates[C]// the 2012 IEEE 28th International Conference on Data Engineering (ICDE). Washington, USA, 2012: 522-533.

[8] GUFLER B, AUGSTEN N, REISER A, et al. Handling data skew in MapReduce[C]// 1th International Conference on Cloud Computing and Service Science. Noordwijkerhout, Netherlands, 2011: 574-583.

[9] ZHOU J S, WANG Q, GAO J. An approach for load balancing Computer in MapReduce via dynamic partitioning[J]. Journal of Computer Research and Development, 2013, 50(S1): 369-377. (in Chinese)
周家帅，王琦，高军. 一种基于动态划分的 MapReduce 负载均衡方法[J]. 计算机研究与发展，2013，50(S1)：369-377.

[10] WANG Z, CHEN Q, LI Z H, et al. An Incremental Partitioning Strategy for Data Balance on MapReduce[J]. Chinese Journal of Computers, 2016, 39(1): 19-35. (in Chinese)
王卓，陈群，李战怀，等. 基于增量式分区策略的 MapReduce 数据均衡方法[J]. 计算机学报，2016，39(1)：19-35.

[11] LI H C, QIN X L, SHEN X. Load Balancing Strategy Based on Pressure Feedback on MapReduce[J]. Computer Science, 2015, 42(4): 141-146. (in Chinese)
李航晨，秦小麟，沈尧. 基于压力反馈的 MapReduce 负载均衡策略[J]. 计算机科学，2015，42(4)：141-146.

[12] KWON Y, BALAZINSKA M, HOWE B, et al. Skew Tune: Mitigating skew in MapReduce applications[C]// the 2012 ACM SIGMOD International Conference on Management of Data. Scottsdale, USA, 2012: 25-36.

[13] LI Q H. Research of Large-scale Data Mining Technologies on MapReduce[D]. Shanghai: Fudan University, 2013. (in Chinese)
李秋虹. 基于 MapReduce 的大规模数据挖掘技术研究[D]. 上海：复旦大学，2013.

[14] FU J, DU Z H. Load Balancing Strategy on Peridical MapReduce Job[J]. Computer Science, 2013, 40(3): 38-40. (in Chinese)
傅杰，都志辉. 一种周期性 MapReduce 作业的负载均衡策略[J]. 计算机科学，2013，40(3)：38-40.

[15] XU Y J. The Research of Parallel Clustering Algorithm of Massive Data in Cloud Computing Environment[D]. Dalian: Dalian Maritime University, 2014. (in Chinese)
许玉杰. 云计算环境下海量数据的并行聚类算法研究[D]. 大连：大连海事大学，2014.