自适应的Spark数据均衡分区方法

何玉林[1,2],吴东彤[1,2],黄哲学[1,2]
(1. 人工智能与数字经济广东省实验室(深圳),广东深圳 518107;2. 深圳大学计算机与软件学院,广东深圳 518060)

摘 要: Spark作为通用的计算引擎,以其简单、快速、可扩展的优势,被广泛地应用于大数据的处理和分析中.然而,Spark默认采用哈希分区或范围分区对数据进行划分,导致其在处理键倾斜分布的数据时,常常出现各分区数据量严重不均衡的问题.诸多优化方法被提出,如迁移分区、贪心分区、反馈分区等,但往往存在数据传输量大、额外计算成本高、运行时间长等问题.为更好地缓解键倾斜分布问题带来的影响,本文提出了一种自适应的Spark数据均衡分区方法.该方法引入了奖惩思想对数据分区过程进行适当调控,同时对于数据量较大的键进行分割,使得各个分区的数据量相对均衡.该方法首先对数据采样并预估键权重.其次,按照键权重对样本数据降序排列,确保所有分区都有初始数据.再次,根据奖惩分配策略,自适应地更新各个分区的分配概率,并将待分配的键指向分配概率最高的分区.对于超过分区容量的键的数据,则分割为多个部分且指向不同分区.在所有样本数据分配完成后,获得自适应分区方案.在实际分区时,对于样本中出现的键对应的数据按照自适应分区方案进行分配;对于未出现的键对应的数据,则按照哈希方法进行分区.最后,通过实验验证,基于新方法设计的自适应均衡分区器(Adaptive Data Balanced Partitioner,ADBP)能够有效缓解键倾斜的负面影响.在真实数据集上,ADBP的WordCount程序总运行时间比自带分区器Hash、Range分别平均缩短了1.51%、29.90%,比现有基于学习自动机的自适应哈希分区器(Learning Automata Hash Partitioner,LAHP)、对倾斜的中间数据块进行拆分合并(Splitting and Combination algorithm for skew Intermediate Data block,SCID)算法、粗粒度放置和细粒度放置(Fined-Coarse Grained Intermediate Data Placement,FCGIDP)算法分别平均缩短了8.12%、21.64%、19.62%.

关键词: 数据倾斜;均衡分区;自适应分区;奖惩分配;Spark
基金项目: 广东省自然科学基金(No.2023A1515011667);深圳市科技重大专项项目(No.KJZD20230923114809020)
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112(2025)08-2764-15
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250348

Adaptive Data Balanced Partitioner in Spark

HE Yu-lin[1,2], WU Dong-tong[1,2], HUANG Zhe-xue[1,2]
(1. *Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ)*, *Shenzhen*, *Guangdong* 518107, *China*;
2. *College of Computer Science and Software Engineering*, *Shenzhen University*, *Shenzhen*, *Guangdong* 518060, *China*)

Abstract: As an open-source computing engine, due to its simplicity, speed and scalability, Spark is widely used in the field of big data processing and analysis. Spark defaults to using hash partitioning or range partitioning to partition data. It often results in severe imbalances in data volume between partitions when processing data with skewed key distributions. Many optimization methods have been proposed, such as migration partitioning, greedy partitioning, feedback partitioning, etc., but often have problems such as large data transmission, high extra computing cost, and long running time. In order to better alleviate the impact of key skew distribution problem, this paper proposes an adaptive Spark data balanced partitioning method, which introduces the idea of reward and punishment to properly regulate the data partitioning process. At the same time, the key with big data volume is properly divided to make the data amount of each partition relatively balanced. After sampling the data and estimating the key weights, the sample data are sorted in descending order according to the key weights, so that all partitions have initial data. Then according to the reward and punishment allocation strategy, the allocation probability of each partition is adaptively updated and the keys to be allocated are directed to the partition with the highest probability. The adaptive data partitioning scheme was obtained after all sample data were allocated. In actual partitioning, the data of keys that appear in the sample are allocated according to the adaptive data partitioning scheme, while the da-

收稿日期:2025-04-30;录用日期:2025-08-18;责任编辑:覃怀银

ta of keys that do not appear are partitioned according to the hash method. The experimental results show that the adaptive data balanced partitioner (ADBP) designed with the new data partitioning method can effectively alleviate the negative impact of key skew. On real data sets, the total running time of WordCount program of ADBP is averagely 1.51% and 29.90% shorter than Spark's own partitioners, i.e., HashPartitioner and RangePartitioner, and averagely 8.12%, 21.64% and 19.62% shorter than the existing partitioners learning automata hash partitioner (LAHP), splitting and combination algorithm for skew intermediate data block (SCID) and fined-coarse grained intermediate data placement (FCGIDP) respectively.

Key words: data skew; balanced partitioning; adaptive partitioning; reward-punishment allocation; Spark

Foundation Item(s): Natural Science Foundation of Guangdong Province (No.2023A1515011667); Major Science and Technology Special Project of Shenzhen (No.KJZD20230923114809020)

1　引言

在传统行业和新兴行业中，大数据技术和平台均不可或缺，用于有效治理和分析每日剧增的生产数据. 主流的大数据平台包括Hadoop[1]、Spark[2]和Flink[3]等. Spark是一个快速、通用、可扩展的计算引擎[4]. 与Hadoop相比，Spark基于内存计算的特点，可以显著减少磁盘I/O操作的次数，提高数据处理速度[5]. 相较于Flink、Kafka[6]等，Spark的生态系统能够提供更全面多样的组件，在批处理和流处理[7]上都具有较好的性能，且支持数据查询、机器学习和图计算等功能[8]. Spark具有更好的数据处理性能和更广泛的应用场景，是当前最热门的分布式平台之一.

现实场景的很多数据存在分布不均的情况，在对这些数据进行分区时，很容易出现各个分区数据量极度不平衡的现象. Spark在对这些数据做处理时，会将运行过程划分为多个阶段(Stage)，每个Stage由多个并行任务(Task)构成，每个Task计算一个分区的数据[9]. 如果各分区的数据量严重不平衡，那么意味着各个Task所需处理的数据量也严重不平衡，即数据倾斜[10]. 此时，在同构环境下，各个Task所需的运行耗时会存在较大差别，分区数据量大的Task会成为决定所在Stage的运行时间的关键，也是限制任务运行效率和资源利用率的瓶颈[11]. 因此，针对数据本身分布不均衡的现象，提出恰当的分区方法使得Spark计算过程中各个分区的数据量保持均衡，能够有效提升任务的运行效率和集群的资源利用率[12].

针对键倾斜问题，现有均衡分区方法主要可以分为迁移分区、贪心分区、反馈分区、改进哈希分区和改进范围分区等. 这些方法在一定程度上能够使得Spark运行过程中各分区数据量相对均衡，缓解了数据倾斜的负面影响，但同时它们也有一些局限性，存在数据传输量大、计算延迟、对采样过度依赖等不足.

为了更好地缓解键倾斜的影响，本文提出了一种自适应的Spark数据均衡分区方法. 该方法在对原始数据采样并预估权重后，按照奖惩分配策略自适应地将键的数据分配到各个分区中，对于数据量过多的键则自适应地分割为多个部分并分配到不同的分区中，实现各个分区的数据量相对均衡. 不同于以往的方法，该方法基于分区状态和已分配数据情况，通过奖惩函数来调节每个分区的分配概率，从而为每个键自适应地选择最合适的分区. 同时，对于超过分区容量的键，该方法根据分区的剩余容量和理想容量自动地将键的数据分割为多个部分，并结合奖罚分配策略，将各个部分分配到恰当的分区中. 本文在Spark环境下基于该方法设计实现了一个自适应均衡分区器(Adaptive Data Balanced Partitioner，ADBP). 由在仿真数据集和真实数据集上的实验结果可知，自适应的Spark数据均衡分区方法能够有效实现键分布极度不平衡的数据在计算过程的均衡分区，能够明显缩短程序运行时间.

2　相关工作

2.1　迁移分区

迁移分区通过建立代价模型或感知模型来检测数据倾斜，将引起倾斜的数据迁移到其他分区. 文献[13]针对Spark SQL(Structured Query Language)中的聚集运算符提出了一种动态执行优化方法，将任务划分为分段任务和窃取任务，当窃取任务完成后，可以主动窃取并处理分段任务的数据，缩短整体运行时间. 文献[14]提出了基于Spark SQL实现的处理系统SrSpark，根据实时可用计算资源自适应地进行节点内和节点间的并行处理，缓解偏斜负载. 文献[15]提出一种自适应倾斜缓解技术，通过事先获取的元数据信息动态地减轻Shuffle过程的数据倾斜. 文献[16]提出了一种新的负载平衡机制，通过线性回归预测Reduce作业的分区大小，利用倾斜检测算法识别出倾斜分区. 文献[17]提出一种动态负载均衡方法，通过数据迁移与数据回迁策略实现最小迁移代价及高效的负载均衡. 文献[18]提出了一种分布式流连接系统FastJoin，通过高效键值选择算法选择尽量少的键值或元组进行迁移. 文献[19]提出了针对异构Spark集群的数据倾斜修正调度策略，将引起倾斜且内存需求明显大于内存的键值重新划分，迁移到不同分区中去.

2.2 贪心分区

根据贪心算法的思想对数据分区过程动态优化，是目前的一大热门方法. 文献[20]提出了一种对倾斜的中间数据块进行拆分合并(Splitting and Combination algorithm for skew Intermediate Data block, SCID)的算法，将超过桶容量的簇进行拆分，通过多次迭代依次填充每一个桶，并使每个桶的数据量大小相对平均. 文献[21]针对数据倾斜问题提出了一种优化分区策略，利用并行聚类采样算法与贪心算法对中间数据进行均衡划分. 文献[22]提出了一个新的Spark数据均衡分区器，以最小化分区倾斜为目标，将数据逐个分配给数据量最少的节点. 文献[23]提出了ReducePartition方法，根据贪心策略将数据均匀地划分到各个分区，并为每个任务分配性能因子最高的执行体. 文献[24]根据数据偏斜度量模型提出粗粒度放置和细粒度放置(Fined-Coarse Grained Intermediate Data Placement, FCGIDP)算法，前者每次将数据分配到最大剩余容量的分区中，后者则依次将数据分配到同一个分区，直到达到该分区的额定容量，顺序转至下一个分区. 文献[25]提出动态均衡分区策略，基于最佳适应算法设计的动态分配原则，在Mapper阶段逐步将切分后的数据块均衡分配到各个分区中. 文献[26]提出了最佳分区搜索算法和一种新的负载计算方法，根据近邻度等信息快速准确地选择分区，通过实时数据分析和预测建模来动态调整资源分配.

2.3 反馈分区

反馈分区是指在分区过程中，基于反馈机制实时调整各个分区或节点的负载量. 文献[27]针对分布式流连接系统提出了一种有状态连接算子的自适应负载均衡策略，利用深度强化学习模型预测流的键频率分布，并提出动态分组算法和基于反馈的资源弹性压缩算法，实时解决热键导致的负载不均衡问题. 文献[28]提出了一种针对流处理作业的动态数据分区策略，动态调整各节点处理的数据量比例，使节点间的数据负载经过多轮反馈调整后处于相对均衡的水平. 文献[29]提出了一种基于学习自动机的自适应哈希分区器(Learning Automata Hash Partitioner, LAHP)，依据环境对自动机的选择行为进行不同的反馈，更新概率向量，使其学会选择最优行为. 文献[30]提出了一种基于反馈调度策略的负载均衡分区算法，该算法可以基于反馈信息为Reduce作业选择最优的分区.

2.4 改进哈希分区和改进范围分区

为了减少额外的计算成本，有些研究选择对原有的哈希分区进行改进. 文献[31,32]均选择为倾斜数据的键添加前缀，避免单个任务处理过多的数据量. 文献[33]提出了一种基于哈希的最佳适应算法，若原哈希分区的剩余容量不满足要求，则根据最佳适应算法为数据重新分配分区. 文献[34]提出了键隔离分区器，对频率最大的k个重键选择原分配策略或加权哈希分区进行分配，对其他非重键使用加权哈希分区进行处理. 文献[35]提出了一种基于业务数据先验信息的Reduce任务负载均衡分区算法，将业务数据中哈希值相同的不同键分散到不同的分区中. 同样地，也有部分研究对原有的范围分区方法提出优化措施. 文献[36]提出了一种轻量级策略来解决数据倾斜问题，基于改进的稀疏索引来定位每个分区的前后位置，在程序语义允许的情况下拆分大键和输出数据的总顺序. 文献[37]针对数据流作业的倾斜问题，提出了一种最优计算分区边界的方法以及一种在语义允许的情况下分裂边界键簇的机制，合理平衡Reduce任务间的负载.

3 自适应分区

3.1 奖惩函数

奖惩函数是指通过奖励或者惩罚对各分区的分配概率进行调节的计算方法，包含奖惩系数、奖惩公式与奖惩结果. 表1为奖惩函数的相关参数及其含义.

表1 奖惩函数的相关参数

参数	含义
N	待分区的数据总量
n	分区的数目
capacity	分区的理想容量
volume	各分区已有数据量的数组
meanVolume	分区的平均已有数据量
totalVolume	分区的总已有数据量
probability	各分区分配概率的数组

当所有分区分配到的数据量相等，即所有数据均分给各个分区时，数据均衡分区效果更佳. 此时，在相同条件下，各个分区对应的Task的运行耗时系统，所属Stage的运行时间最短. 在自适应分区方法中，将这一理想状态下的分区数据量作为各分区的理想容量. 具体地，理想容量为待分区的数据总量与分区数的相除结果，即

$$\text{capacity}=\frac{N}{n} \tag{1}$$

分区的平均已有数据量是指当前所有分区的已有数据量的平均值，即

$$\text{meanVolume}=\frac{1}{n}\times\text{totalVolume} \tag{2}$$

$$\text{totalVolume}=\sum_{i=1}^{n}\text{volume}(i) \tag{3}$$

当某分区的已有数据量超过平均已有数据量时，

继续往该分区分配数据可能会引发或者加剧分区间数据量倾斜的情况；相反地，当分区的已有数据量远远少于平均已有数据量时，将数据分配到该分区有益于缩短数据量差距，降低各分区已有数据量的方差. 因此，自适应分区方法将当前分区的平均已有数据量作为奖惩函数的判断条件和调节因子.

为了更好地体现各分区状态，自适应分区方法使用均值差值来衡量各分区已有数据量与分区平均已有数据量的差距，表示为disWithMean，具体计算公式为

$$\text{disWithMean}=\frac{|\text{volume}(i)-\text{meanVolume}|}{\text{capacity}} \tag{4}$$

其中，i为分区索引号；$|\text{volume}(i)-\text{meanVolume}|$为第$i$个分区的已有数据量与当前分区平均数据量的绝对差额. 将绝对差额与理想容量的比值作为后续调节分区的分配概率的系数之一，从而将当前各分区状态与均衡分区过程联系起来.

奖惩函数将各分区的状态分为以下3种，并根据不同的状态设置不同的分配概率更新公式，其中，i为分区索引号，取值范围为$[1,n]$.

（1）当分区已有数据量超过理想容量，即$\text{volume}(i)\geqslant\text{capacity}$时，该分区的分配概率更新为0：

$$\text{probability}(i)=0 \tag{5}$$

在此情况下，继续向该分区增加数据，不仅扩大了该分区与其他轻负载分区的数据量差距，还拉大了最终分区效果与理想均衡分区效果的差距，导致数据倾斜影响加重，故后续都不适合再分配数据给该分区.

（2）当分区已有数据量超过平均已有数据量，即$\text{volume}(i)\geqslant\text{meanVolume}$时，该分区的分配概率适当降低：

$$\text{penalty}(i)=\text{penaltyCoe}\times\left(1+\frac{\text{totalVolume}}{N}\right)\times\text{disWithMean}(i) \tag{6}$$

$$\text{probability}(i)=\text{probability}(i)\times\left[1-\text{penalty}(i)\right] \tag{7}$$

其中，penaltyCoe为惩罚系数，取值范围为(0,1)；$\text{penalty}(i)$为第i个分区的惩罚幅度. 在此情况下，理论上不应在下一次分配中将数据分配给该分区. 当前已完成分配的数据越多，或者当该分区的已有数据量的均值差值越大时，越不适合分配，需要的惩罚幅度越大，相反则惩罚幅度越小.

（3）当分区已有数据量少于平均已有数据量，即$\text{volume}(i)<\text{meanVolume}$时，该分区的分配概率适当提高：

$$\text{reward}(i)=\text{rewardCoe}\times\left(1+\frac{\text{totalVolume}}{N}\right)\times\text{disWithMean}(i) \tag{8}$$

$$\text{probability}(i)=\text{probability}(i)\times\left[1+\text{reward}(i)\right] \tag{9}$$

其中，rewardCoe为奖励系数，取值范围为(0,1)；$\text{reward}(i)$为当前第i个分区的奖励幅度. 在此情况下，在下一次分配中适合将数据分配给该分区，可以适当进行奖励. 奖励幅度的大小机制与惩罚幅度相似.

在奖励函数中，调控奖惩幅度的参数不仅有奖惩系数和分区已有数据量的均值差值，还有当前已完成分配的数据量占全部数据的比例. 已分配的数据越多，表明后续待分配的数据越少，各分区数据量的可调控范围越小，即平衡各分区数据量的需求越紧迫. 在这种状态下，应优先将数据分配到数据量较少的分区，尽量避免向数据量较多的分区添加数据，从而加大奖惩力度，缩小各分区数据量的差距. 基于此，自适应分区方法将已分配数据占总数据的比例作为参数之一，调节每轮分配后各分区分配概率的奖惩幅度，以更好地实现最终的数据均衡分区.

在每一次对分配概率进行适当调整后，奖惩函数还需要对更新后的分配概率进行标准化处理，即

$$\text{probability}(i)=\frac{\text{probability}(i)}{\sum_{u=1}^{n}\text{probability}(u)} \tag{10}$$

标准化处理是为了将各分区的分配概率调整到同一尺度下，即分配概率的总和为1，有利于提高后续均衡分配的准确度和稳定性.

奖惩函数针对不同状态对分区的分配概率做适当调节，并对其进行标准化处理. 算法1是奖惩函数的伪代码.

3.2　自适应分区方案

本文所提出的自适应数据均衡分区方法，基于奖惩分配策略和热键分割方法，对样本中出现的键进行分析和计算，自适应地将数据均衡分配到各个分区中，获得数据分区方案. 奖惩分配策略是指按序获取待分配的下一个键，选择当前分配概率最高的分区作为意向分区，将该键的数据分配到意向分区中. 同时，根据当前分配情况，通过奖惩函数适当调节各个分区的分配概率，重复这一步骤直至所有键的数据都完成分配. 热键分割操作是指在执行奖惩分配策略时，当键的权重超过意向分区的剩余容量时，将该键的数据分割为两部分，放置到不同的分区中. 图1展示了获取自适应分区方案的主要流程.

为了更好地阐明获取自适应分区方案的流程，表2展示了自适应分区方案的相关参数及其含义.

自适应获取分区方案的具体步骤如下：

（1）对样本中的键按照权重降序排列. 在采样并估算样本中的键在全部数据中的权重后，获得数组$\text{KW}=\left\{(\text{key}_1,\text{weight}_1),(\text{key}_2,\text{weight}_2),\cdots,(\text{key}_{N_c},\text{weight}_{N_c})\right\}$，对

算法1 奖惩函数

输入: 分区理想容量 capacity
各分区已有数据量的数组 volume
各分区分配概率的数组 probability

输出: 更新后的各分区分配概率 probability

1. totalVolume = $\sum_{i=1}^{n}$volume(i)
2. meanVolume = totalVolume/n
3. For $i=1,2,\cdots,n$ DO
4. IF volume(i)≥capacity DO
5. probability(i)=0
6. ELSE IF volume(i)≥meanVolume DO
7. 惩罚,probability(i)适当降低
8. ELSE DO
9. 奖励,probability(i)适当增高
10. END IF
11. END FOR
12. probabilitySum = $\sum_{i=1}^{n}$probability(i)
13. FOR $i=1,2,\cdots,n$ DO
14. probability(i)= $\frac{\text{probability}(i)}{\text{probabilitySum}}$
15. END FOR

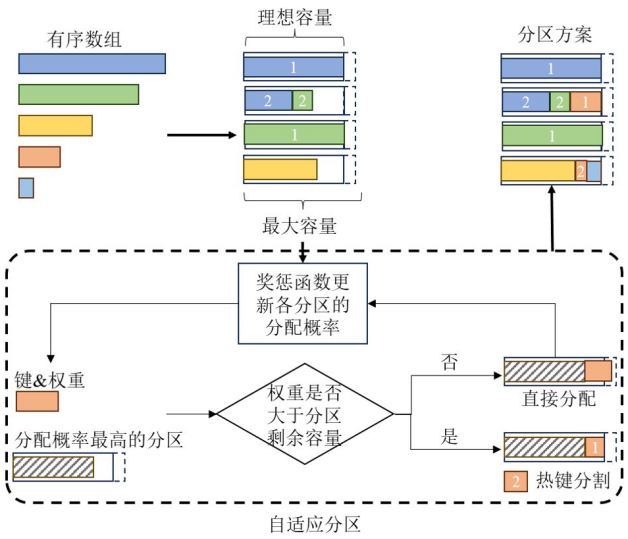


图1 获取自适应分区方案的主要流程

表2 自适应分区方案的相关参数

参数	含义
N_c	样本中不同键的个数
key$_j$	样本中的第j个键
weight$_j$	样本中第j个键的权重
remain	各分区剩余容量的数组

数组KW按照权重降序操作,获得有序数组$\text{KW}_{\text{scorted}}=\left\{(\text{key}'_1,\text{weight}'_1),(\text{key}'_2,\text{weight}'_2),\cdots,(\text{key}'_{N_c},\text{weight}'_{N_c})\right\}$,键的权重在后续操作中被看作键对应的数据量.

(2)初始化奖惩分配策略的相关变量. 根据式(1)计算分区的理想容量capacity. 将各个分区的已有数据量初始化为0,剩余容量初始化为分区可容忍的最大数据量,分配概率设置为初始均值,具体如下:

$$\text{volume}(i)=0 \tag{11}$$

$$\text{remain}(i)=\text{capacity}\times\text{tolerance} \tag{12}$$

$$\text{probability}(i)=1/n \tag{13}$$

其中,i为分区的索引号,取值范围为$[1,n]$. 分区可容忍的最大数据量为理想容量capacity与用户定义的分区容忍度tolerance的乘积. 在实际场景下,各个分区的最终数据量很难恰好达到理想容量. 因此,自适应数据均衡分区方法引入了分区容忍度这一变量,用于表示用户能够接受的分区数据量倾斜程度,取值范围为(1,2). 将分区可容忍的最大数据量看作分区的最大容量,在后续获取分区方案时,不允许分区的数据量超过最大容量.

(3)令每个分区都有原始数据. 将前几个键的数据分配到各个分区中,确保每个分区都有原始的键数据,以获得各个分区的原始分配概率,提高后续使用奖惩分配策略的有效性和准确性. 按序遍历有序数组中待分配的键,将其按规则分配到空分区中,超过分区最大容量的部分则执行热键分割操作,并分配至合适的分区. 重复该步骤,直至所有分区都有数据. 假定当前空分区的索引为e,按序获取待分配的键和权重$(\text{key}_j,\text{weight}_j)$,其中$j$的取值范围为$[1,n]$. 规则如下:

①若weight$_j$小于等于空分区的最大容量,则将该键的数据全部分配给当前空分区,更新分区方案和相关变量.

②若weight$_j$大于空分区的最大容量,则将该键的数据进行分割,一部分的大小为capacity,分配到当前空分区,另一部分的大小为weight$_j$−capacity,重新放入待分配数组中,等待下一次分配. 更新分区方案和相关变量.

接着,判断此时是否还有空分区,若有,指向下一个空分区和下一个待分配键,$e=e+1$,$j=j+1$,重复步骤(3).

(4)更新各分区的分配概率. 调用第3.1节介绍的奖惩函数,以各分区的已有数据量为依据,对各分区的分配概率进行适当的奖励或者惩罚,调整下一轮分配中将键分配到不同分区的概率,当前数据量少的分区的概率大一点,数据量大的分区的概率小一点.

(5)按照奖惩分配策略和热键分割操作分配有序数组中剩余的键的数据,更新分区方案. 假定待分配的下一个键和权重为$(\text{key}_j,\text{weight}_j)$,获取分配概率最高

的分区的索引并记为h,其中$1\leqslant j\leqslant N_c$,$1\leqslant h\leqslant n$,$N_c$为样本中不同键的数目. 那么该键分配到分区$h$的数据量会有以下2种情况:

①若键的权重小于等于分区的剩余容量,即$\text{weight}_j\leqslant\text{remain}(h)$,则直接将该键的全部数据分配到分区$h$中,更新分区的已有数据量以及分区方案.

②若键的权重大于分区的剩余容量,即$\text{weight}_j>\text{remain}(h)$,则对该键执行热键分割操作. 一部分放入分区$h$,大小为$\text{wp1}=\text{capacity}-\text{volume}(h)$,使分区$h$的数据量到达理想容量,更新分区方案. 另一部分重新放入待分配数组中,大小为$\text{wp2}=\text{weight}_j-\text{wp1}$,等待下一次分配.

(6)判断是否完成所有键的分配. 若待分配数组中所有元素都已经遍历,表示样本中所有键的全部数据都已完成分配,获得自适应分区方案;若待分配数组中还有键未完成分配,则$j=j+1$,重复步骤(4)和步骤(5).

自适应分区方法通过奖惩分配策略和热键分割操作相结合,对各分区的分配概率进行奖励或惩罚,对键的数据进行分割或完整分配,为每个键的数据选择恰当的分区,自适应地平衡各个分区分配到的数据量,最终获得各分区数据量相对均衡的自适应分区方案. 算法2是获取自适应分区方案的伪代码.

由算法2可知,生成的自适应分区方案中包含键、分区号以及对应分配的权重,主要目的是标明发生分割的键分配到不同分区中的数据量,以便后续实际分区阶段也可以对相应的键进行分割. 为了在实际分区阶段更准确地将分割键的不同数据分配到正确的分区中,我们将分区方案中的权重转换为该键总权重的累计占比,并将同一键的多个分区映射转换为链表. 具体的伪代码如算法3所示.

键倾斜是指在Shuffle阶段,部分键的数据量远远超过其他键的数据量,即热键的数据量占了全部数据的绝大比例. 如果遵循原有的分区方法,相同键的全部数据分配到同一个分区中,那么热键对应的分区的数据量就会远远大于其他分区. 为了缓解键倾斜的负面影响,需要将热键的数据划分成多个部分,并将划分后的热键数据和未划分的键数据一同均衡地分配到各个分区中. 基于此,自适应数据均衡分区方法使用了奖惩分配策略和热键分割操作相结合的方式,以实现存在键倾斜的数据的均衡分区.

如何对热键进行分割是面临的挑战之一. 判断哪些键需要分割,分割为几个部分,各个部分的数据量比例是多少,不同部分处理的先后顺序怎么安排等,这些因素都直接影响最终的数据均衡分区效果. 针对这些问题,自适应分区方法采取自适应键分割的方式,为每个分区设置理想容量和用户可容忍的最大容量,每次及时更新分区的已有数据量和剩余容量. 当键的数据量超过分区的剩余容量时,该键需要先分割为2个部分,一部分的数据量为理想容量与分区已有数据量的差值,放入当前分区中;另一部分的大小则为原数据量减去前一部分的数据量的结果,放入待分配数组中,等待下一次分配. 通过该方式,自适应地确定键是否进行分割以及各个分割点,确保分割后的结果是当前分区状态的局部最优解,也确保后续分区数据量的倾斜程

算法2　获取自适应分区方案

输入: 预估的键及权重数组KW

输出: 自适应分区方案scheme

```
1. KW_sorted←KW按照权重降序
2. 计算理想容量capacity
3. 初始化volume、probability的每个元素为0
4. 初始化remain的每个元素为capacity × tolerance
5. j = 1
6. WHILE 存在空分区 e DO
7.     IF weight_j ≤ remain(e) DO
8.         scheme.add(key_j, e, weight_j)
9.         更新相关变量
10.        j = j + 1
11.    ELSE DO
12.        //执行热键分割操作
13.        scheme.add(key_j, e, capacity)
14.        更新相关变量
15.        wp2 = weight_j − capacity
16.        将(key_j, wp2)重新放入KW_sorted
17.    END IF
18.    空分区数目减1
19. END WHILE
20. WHILE j ≤ Nc DO
21.    //执行奖惩分配策略
22.    调用奖惩函数,更新各分区的分配概率
23.    选择分配概率最大的分区 h
24.    IF weight_j ≤ remain(h)的剩余容量 DO
25.        scheme.add(key_j, h, weight_j)
26.        更新相关变量
27.        j = j + 1
28.    ELSE DO
29.        //执行热键分割操作
30.        wp1 = capacity − volume(h)
31.        wp2 = weight_j − wp1
32.        scheme.add(key_j, h, wp1)
33.        更新相关变量
34.        将(key_j, wp2)重新放入KW_sorted
35.    END IF
36. END WHILE
```

算法3 分区方案汇总转换

输入: 分区方案 scheme

输出: 汇总后的分区方案 $scheme_{final}$

```
1. 创建 schemeByKey,键为字符串类型,值为链表类型
2. FOR sch←scheme DO
3.     IF schemeByKey 已经包含 sch.key DO
4.         schemeByKey(sch.key).add((sch.partition, sch.weight))
5.     ELSE
6.         schemeByKey 中创建新键 sch.key
7.         schemeByKey(sch.key).add((sch.partition, sch.weight))
8.     END IF
9. END FOR
10. FOR sbk←schemeByKey DO
11.     wsum←SUM(sbk.weight)
12.     FOR pw←sbk.partition, sbk.weight DO
13.         ratio←sbk.partition, sbk.weight
14.         schemeByKey(sch.key).add((sch.partition, sch.weight))
15.     END FOR
16. END FOR
```

度在用户可容忍范围内.

如何均衡分配各个键的各部分数据是研究的核心挑战之一. 每次为当前待分配的键所选择的分区,其结果既影响后续键数据的分配,又关乎热键分割操作的执行. 如果选择的意向分区剩余容量非常少,那么当前键需要执行分割操作的可能性就会增高,且键分割后该部分的数据量也会非常少,这可能会增加后续不必要的计算成本. 如果选择的意向分区的已有数据量已经大大超过平均已有数据量,继续分配数据会加大分区之间数据量的不平衡,降低数据均衡分区效果. 因此,为了选择一个合适的分区,自适应分区方法引入了奖惩分配策略作为选择意向分区的依据. 根据奖惩系数、奖惩力度以及奖惩行为的有效结合,自适应地调节各分区的分配概率. 通过判断当前分区状态,衡量各分区是否适合继续分配数据,从而确定对各分区的分配概率做出奖励或者惩罚操作. 分配概率反映了将数据分配到不同分区的合理性. 分配概率越高,表明分区的数据量较少且适宜分配,反之则表明不适合分配数据,故每次都选择分配概率最高的分区作为意向分区,自适应地实现不同键的数据的均衡分区.

每一个 Task 只计算一个分区的数据,分区间的数据或者信息不会共享. 对于聚合类算子,热键分割操作会导致相同键的数据被分散到不同分区中,由不同 Task 进行处理. 此时,Task 只能计算获得各自分区的局部聚合结果,需要多一个步骤对局部聚合结果进行汇总. 然而,由于前一次聚合已经大大减少了数据量,局部聚合结果与最终结果之间只存在少数分割的键的差异,故第二次聚合所需要处理的数据量非常少,所需耗时几乎可以忽略不计. 即二次聚合所增加的成本对于总运行时间的影响较小,而热键分割操作却可以显著缩短局部聚合阶段的执行时间,从理论上看,热键分割操作利大于弊,具有一定的合理性和科学性.

4 ADBP

根据前文所阐述的自适应分区方法原理,本节设计并实现了 ADBP. 其主要由 4 个模块构成,分别是采样与权重估计、自适应分区、分区方案汇总转换和实际分区,结构如图 2 所示.

在采样与权重估计模块中,通过蓄水池算法[38]对 Shuffle 阶段的中间数据进行采样,统计样本中每个键的出现频率,接着根据采样比例反推每个键在全部数据中所占的权重,获得由键和权重组合作为元素的待分配数组,并将数组中的元素按照键的权重降序排列.

自适应分区模块由奖惩分配策略和热键分割操作 2 个部分组成,可得到由键、分区号和分配权重三者组合作为元素的分区方案. 其中,奖惩分配策略和热键分割操作不是完全独立的关系,而是交替执行. 这一模块的大致过程如下:首先,奖惩分配策略更新各分区的分配概率和确定当前键的意向分区;其次,判断意向分区的剩余容量与当前键数据量的大小关系,确定是否执行热键分割操作;最后,重复上述步骤,直至样本中的所有键的全部数据都分配完成.

在自适应分区模块所获取的分区方案中,由于热键分割操作,可能存在同个键的不同数据被分配到不同分区的情况. 为了便于后续实际分区过程的识别,在分区方案汇总转换模块中,将原分区方案中相同键的元素汇总,计算相同键中分配到不同分区的数据量的累计占比,并转换成链表形式,节点元素为分区和对应累计占比的组合,获得最终的自适应分区方案. 此时,链表中每一个节点的累计占比与其上一个节点的差值,表示该键的对应数据分配到该节点所指向分区的数据比例,差值越大,表明该键分配到该分区的数据量越多.

在实际分区中,根据转换后的自适应分区方案对 Shuffle 阶段的每一个数据分配分区. 对于不包含在分区方案中的键的数据,按照哈希分区方法分配分区. 反之,先获取分区方案中该键对应的链表,并生成一个范围为 (0, 1) 的随机数,依次比较随机数和链表节点中的累计占比值,当随机数小于访问节点的比值时,将数据分配到该节点指向的分区.

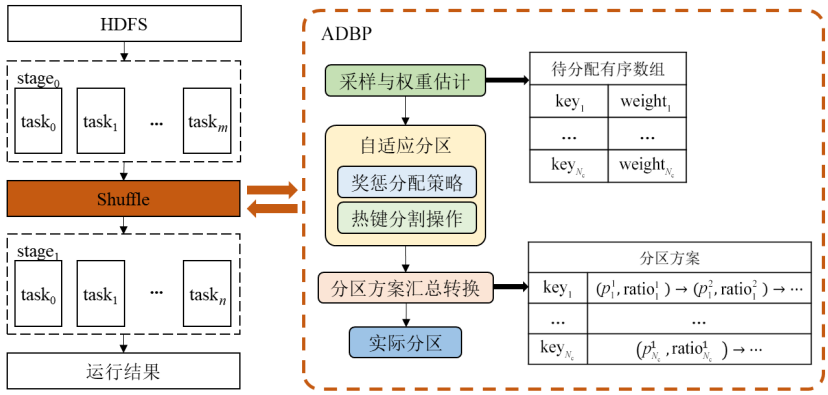


图2　ADBP的主要结构

5　实验与分析

5.1　实验设置

实验环境为5个节点构成的集群,由Centos Linux 7、Spark 2.4.0、Scala 2.11.12和Hadoop 2.7.4搭建而成. 在运行程序时,统一设置Executor的个数为8个,每个Executor有4 GB内存和4个核,Driver的内存大小是16 GB,最多可用的Executor数目是8个. 同时,在执行倒排索引程序时,将序列化缓冲区的最大容量增大到256 MB. 由于在分布式环境下,程序的执行耗时有一定的不稳定性,本文通过重复运行5次并取平均值作为实验结果.

实验过程选取了2个应用程序,词频统计(WordCount)和倒排索引(InvertedIndex). 实验数据集包括仿真数据集和真实数据集,具体信息如表3所示. 仿真数据集是符合标准Zipf分布[39]的键倾斜数据集,分别包含7个子数据集,每个子集有1 000万条数据,数据的键分布是指数为γ的Zipf分布,指数范围为[0.5,3.5],以0.5为单位间隔. γ越大,表明键分布的倾斜程度越大. 真实数据集是美国政府在官方网站发布的航空数据集(Bureau of Transportation Statistics, BTS)[40],选取了2018—2023年的全部数据,按照年份划分为不同子数据集.

表3　不同实验及数据集的具体信息

实验类别	实验程序	数据集	子集个数	平均大小/MB	描述
可行性验证	WordCount	Zipf	3	30.3	标准Zipf分布的整数数据
	InvertedIndex	Zipf	3	82.1	标准Zipf分布的键值数据对,键为字符串,值为整数
有效性验证	WordCount	Zipf	7	30.6	标准Zipf分布的整数数据
	InvertedIndex	Zipf	7	81.2	标准Zipf分布的键值数据对,键为字符串,值为整数
	WordCount	BTS	6	481.7	2018—2023年的US航班数据

5.2　评估指标

均衡分区的目标是:在Shuffle过程中,键分布严重倾斜的数据能相对均衡分配到各个分区中,缩短下一个Stage中各个Task运行时间差距,提高整个程序的运行效率. 鉴于该目标,实验设置了以下4个指标来评估分区器的性能:

(1)总运行时间. 即程序运行所需的总时间,均衡分区方法的最终目标是缩短任务的运行时间,提高运行效率.

(2)获取分区方案时间. 即采样到获取最终分区方案这一过程所需的时间,是均衡分区器的额外耗时之一. 这个时间值越小,表示额外增加的成本越低,间接反映了分区器的性能.

(3)下一个Stage运行时间. 即在Shuffle操作后,执行下一个Stage所需的运行时间. 这一指标间接展现了分区器的均衡分区效果,该值越小,表示均衡分区效果越好.

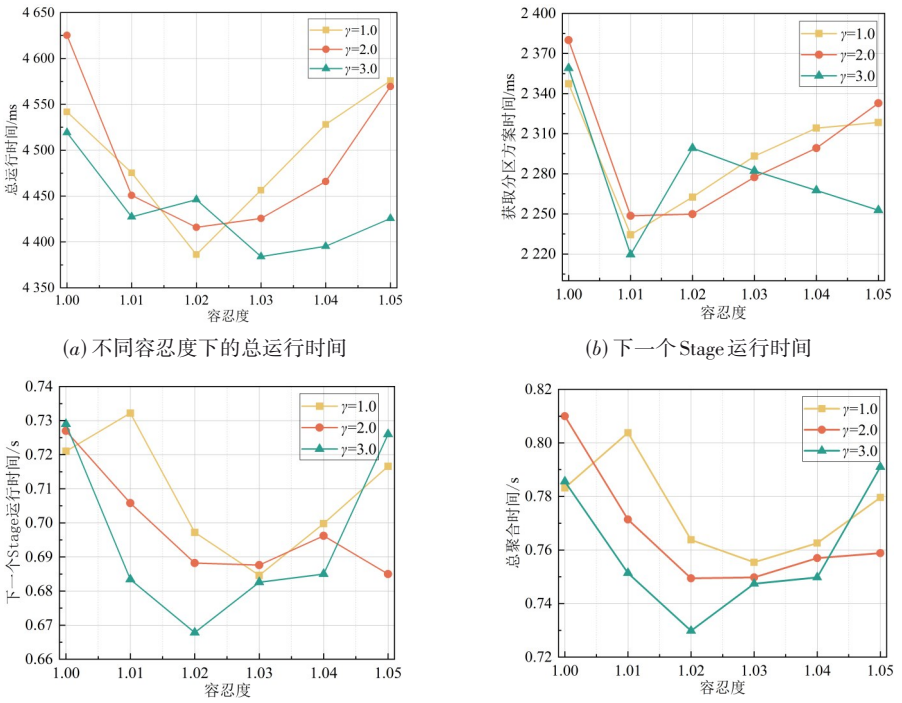
(4)总聚合时间. 即Shuffle后获得最终结果所需的时间,即2次聚合所需的总时间. 分区器可能会需要二次聚合,这一指标可以体现分区器的均衡分区效果是否真实且有效.

5.3　可行性验证

为了验证自适应数据均衡分区方法的可行性,本节在Zipf数据集上测试了不同容忍度对ADBP性能的影响. 同时,为了验证在不同键倾斜程度下,容忍度对ADBP的数据均衡分区效果的影响是否一致,本节将分别在γ=1.0、γ=2.0和γ=3.0的Zipf数据集上进行可行性验证实验. 实验结果包含4个子图,分别展示了程序的

总运行时间、获取分区方案时间、下一个Stage运行时间和Shuffle后的总聚合时间.

图3是ADBP在WordCount程序上的可行性验证结果. 横坐标是ADBP的容忍度取值,范围为[1.00,1.05],以0.01为相邻间隔.可以看出,在不同γ值的数据集上,ADBP的容忍度取值对WordCount程序的影响相似,各个指标的变化趋势大体一致. 当容忍度的取值小于1.01时,ADBP在WordCount程序上的各个评估指标都随着容忍度的增大而减小;当容忍度的取值在1.01~1.03时,ADBP在不同γ值数据集上的各个评估指标存在略微波动,但均处于一个较低的范围;当容忍度的取值大于1.03时,ABDP在WordCount程序上的各个评估指标都随着容忍度值的增大而增大,包括程序的总运行时间和额外增加的时间成本,即性能不断降低. 其中,图3(*b*)中,当容忍度值大于1.02时,ADBP在γ=3.0的数据集上获取分区方案的时间逐渐减小,与其他数据集的变化存在一定差异,但是在其他评估指标上,γ=3.0的数据集变化趋势和其他数据集基本上一致. 因此,该部分差异在正常可接纳的范围内,本节将其看作实验的误差,忽略不计.综上所述,当容忍度取值在[1.01,1.03]时,ADBP在WordCount程序上的性能相对更好.



(*a*) 不同容忍度下的总运行时间 (*b*) 下一个Stage运行时间

(*c*) 不同容忍度下的下一个stage运行时间 (*d*) 不同容忍度下的总集合时间

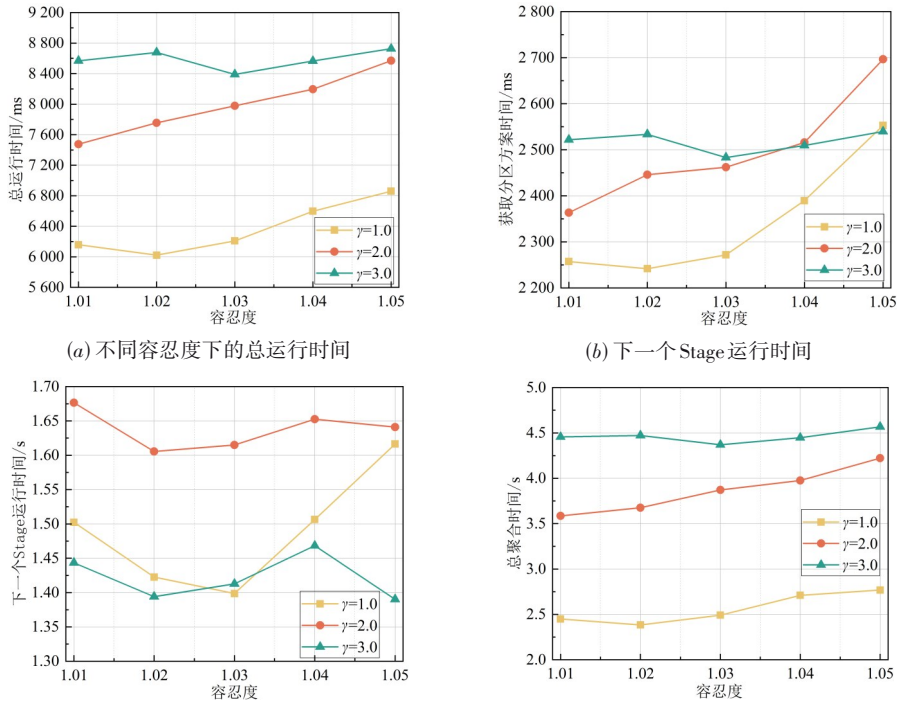
图3 ADBP在WordCount程序上的可行性验证结果

图4展示了ADBP在InvertedIndex程序上的可行性验证运行结果. 其中,横坐标为容忍度的大小,取值范围为[1.01,1.05],增量为0.01. 可以看出,在4个子图中,不同γ值数据集的折线趋势都基本相同,即在不同键倾斜程度下,容忍度取值大小对ADBP在InvertedIndex程序中的性能的影响几乎相同. 在InvertedIndex程序中,ADBP在程序总运行时间、获取分区方案时间和总聚合时间这3个评估指标上均随着容忍度取值的增大而增大,而下一个Stage运行时间的最低值在容忍度为1.02和1.03处波动.从总体上看,当容忍度位于[1.01,1.02]时,ADBP在InvertedIndex程序中的均衡分区效果相对更好.

从2个程序的可行性验证结果可以看出,当容忍度取值在1.01左右时,ADBP的均衡分区效果都相对较好,故后续有效性验证实验中,ADBP的容忍度值都设为1.01.

5.4 有效性验证

为了证明ADBP在键倾斜场景中的优越性,本节对ADBP进行有效性验证,将ADBP与HashPartitioner[41](简写为HASH)、RangePartitioner[41](简写为RANGE)、LAHP[29]、SCID[20]、FCGIDP[24]进行性能对比,在WordCount程序和InvertedIndex程序上运行不同数据集并,比较不同分区器的评估指标结果. 其中,HASH和RANGE是Spark自带的2个默认分区器,LAHP、SCID和FCGIDP是目前较为先进的、有代表性的、面向键倾斜的数据均衡分区器. 由于HASH和LAHP都是一次性分区器,没有获取分区方案这一步骤,故在结果图中两者的获取分区方案时间均为0,没有显示. 同样地,实验



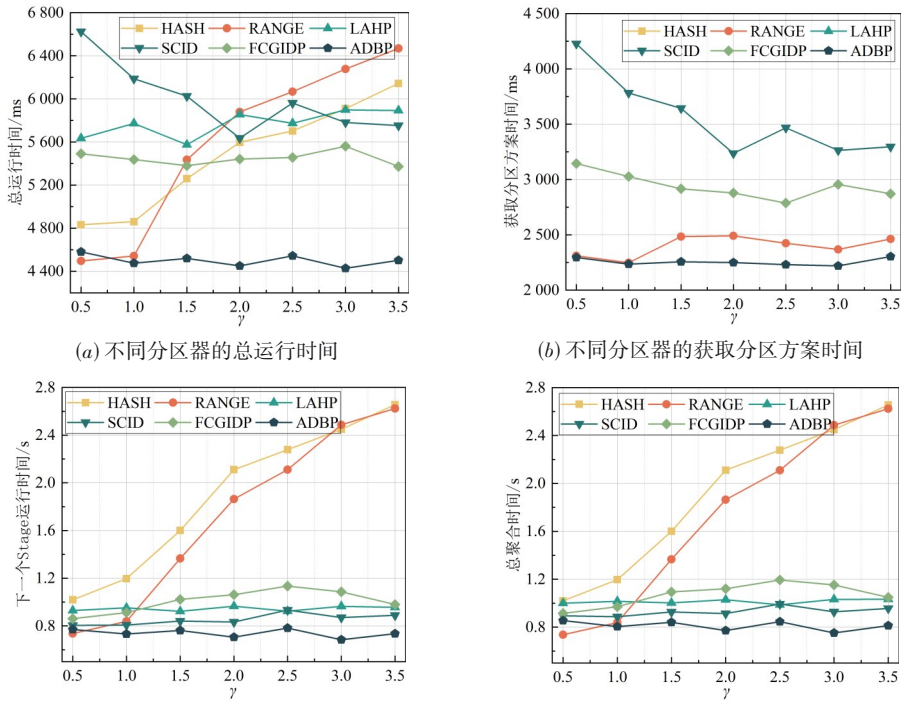
(*a*) 不同容忍度下的总运行时间　(*b*) 下一个Stage运行时间

(*c*) 不同容忍度下的下一个stage运行时间　(*d*) 不同容忍度下的总集合时间

图4　ADBP在InvertedIndex程序上的可行性验证结果

结果各有4个子图,分别对应4个评估指标,即程序的总运行时间、获取分区方案时间、下一个Stage运行时间和总聚合时间,单位依次为ms、ms、s和s.

图5展示了在WordCount程序上统计不同γ值的Zipf数据集调用不同分区器的运行结果.横坐标都为分布指数γ,范围为[0.5,3.5],增量为0.5. 由图5(*a*)~图5(*d*)可知,在WordCount程序中,随着γ值的增大,HASH和RANGE的运行时间不断增加,而LAHP、SCID、FCGIDP



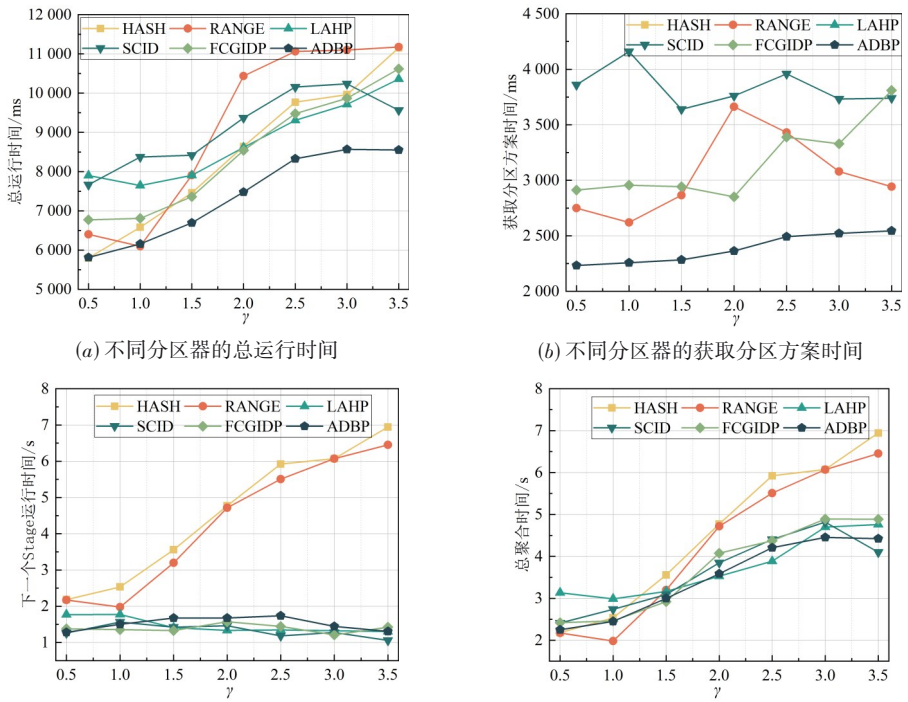
(*a*) 不同分区器的总运行时间　(*b*) 不同分区器的获取分区方案时间

(*c*) 不同分区器的下一个Stage运行时间　(*d*) 不同分区器的总聚合时间

图5　不同分区器在Zipf数据集上执行WordCount程序的运行结果

和ADBP的运行时间则仅在一定区间内浮动. 同时,在γ>1.0的数据集中,这4个均衡分区器的下一个Stage运行时间和总聚合时间都远远小于同一γ值的HASH和RANGE的对应指标. 明显地,ADBP的所有运行时间指标均最低且最稳定. 在总运行时间方面,ADBP比LAHP平均低21.98%,比SCID平均低24.78%,比FCGIDP平均低17.40%;在总聚合时间方面,ADBP比LAHP、SCID和FCGIDP分别平均少了19.90%、12.50%和23.50%. 这一现象说明,4个均衡分区器在WordCount程序上都一定程度地实现了Shuffle阶段的数据均衡分区,并且ADBP的数据均衡分区效果相对更明显更有效,性能较优. 在图5(b)中,RANGE、SCID、FCGIDP和ADBP的获取分区方案时间都各自限定在一个范围内上下波动,且ADBP的获取分区方案时间较前三者的分别平均降低了5.85%、36.15%、23.19%,即说明ADBP相对其他分区器所需额外增加的成本更低.

图6展示了在InvertedIndex程序执行不同分区器的运行结果,使用不同倾斜程度的Zipf数据集,横纵坐标的意义与图5相同. 由图6(a)和图6(d)可知,随着γ值的增加,即数据集的键倾斜程度增加,不同分区器的程序总运行时间和总聚合时间均在增加,区别在于HASH和RANGE的增幅较大,而其他分区器的时间增长较为缓慢. 其中,ADBP的总运行时间和总聚合时间基本最小,增长速度也最低. 具体地,ADBP的总运行时间比LAHP、SCID、FCGIDP的总运行时间分别平均降低了16.33%、19.46%和12.86%;在总聚合时间这一指标中,ADBP相比LAHP平均减少了7.68%,比SCID平均减少4.44%,比FCGIDP平均减少5.56%. 此外,在图6(c)中,使用HASH和RANGE分区的程序,下一个Stage运行时间都随着键倾斜程度的增大而不断增加,而使用LAHP、SCID、FCGIDP和ADBP的程序,其下一个Stage运行时间则一直1~2 s波动,远远低于HASH和RANGE. 从图6(d)来看,不同分区器的获取分区方案时间在一个范围内波动,其中RANGE和FCGIDP的波动范围较大,SCID和ADBP的波动范围较小,四者的平均值大小关系为SCID>FCGIDP>RANGE>ADBP. 具体来讲,ADBP的获取分区方案时间比RANGE、SCID、FCGIDP的获取分区方案时间分别平均减少了17.58%、37.68%、24.33%. 综合4个评估指标,与其他分区器相比,ADBP在InvertedIndex程序上的性能相对更佳,数据均衡分区效果更明显且较为稳定,额外增加的时间成本更低且较为稳定.



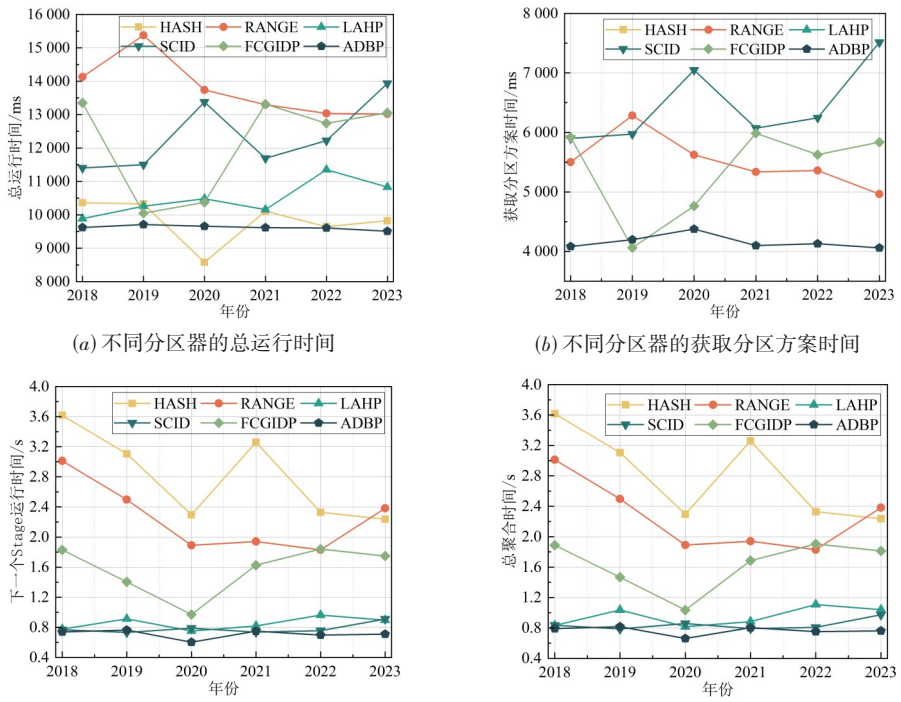
(a) 不同分区器的总运行时间　　(b) 不同分区器的获取分区方案时间

(c) 不同分区器的下一个Stage运行时间　　(d) 不同分区器的总聚合时间

图6　不同分区器在Zipf数据集上执行InvertedIndex程序的运行结果

图7展示了在WordCount程序中调用不同分区器的运行结果,使用BTS数据集. 横坐标是年份,即2018—2023年. 在该实验中,WordCount程序的功能是统计BTS数据集中不同年份、不同延迟时间的US航班个数. 一般情况下,US航班会准时或者延迟几分钟出发和到达,每年延迟时间低于30 min的US航班个数会远远大于延迟时间高于30 min的US航班个数,于是会出现以延迟时间为键的数据在Shuffle阶段发生严重倾斜的情

况,这一场景匹配本章所研究的键倾斜问题. 从图7来看,在不同年份的子集中,其他分区器在不同评估指标上各有优劣,但ADBP的不同时间指标基本都是所有分区器中最小的,除了2020年特殊时期. 在总运行时间上,ADBP比HASH、RANGE、LAHP、SCID和FCGIDP依次平均少了1.51%、29.90%、8.12%、21.64%和19.62%;在总聚合时间上,ADBP依次平均降低了72.02%、65.28%、18.94%、8.59%和51.59%. 同时,从额外增加的成本来看,ADBP的获取分区方案时间比RANGE低了24.26%,比SCID低了35.12%,比FCGIDP低了20.71%. 由此证明,在实际应用场景中,ADBP较现有的分区器拥有相对更好的性能.



(*a*) 不同分区器的总运行时间　(*b*) 不同分区器的获取分区方案时间

(*c*) 不同分区器的下一个Stage运行时间　(*d*) 不同分区器的总聚合时间

图7　不同分区器在BTS数据集上执行WordCount程序的运行结果

由上述实验结果可知,在不同的应用程序及不同的数据集中,ADBP均能得到良好的均衡分区效果. 相较于其他分区器,ADBP不仅使得倾斜数据能够相对均衡地分配到各个分区中,还缩短了任务的总运行时间,提高了运行效率,且没有产生过多的额外成本. 这主要得益于ADBP中的自适应分区方法,即奖惩分配策略和热键分割操作. 不同于其他分区方法中单一的分配策略,奖惩分配策略根据奖惩机制,基于当前分区状态对各个分区给予奖励或者惩罚,每次都选择分配概率最高的分区作为意向分区,从而实现动态平衡各个分区的数据量. 热键分割操作是在待分配键的数据量超过意向分区的剩余容量时进行,它将同一热键的数据尽可能地划分为少数份,以减少后续二次聚合的成本. 而且,相较于其他方法中将数据全部打散,或者预先将数据量较大的键划分为多个部分,ADBP的热键分割操作能够针对性地划分数据,减少不必要的额外成本. 从某种角度来说,自适应分区方法可以看作将反馈分区和贪心分区的核心思想相融合的结果,具有一定的合理性和流程可行性:奖惩分配策略保留了反馈分区的动态调整,热键分割操作保留了贪心分区的局部最优策略,这些优势使得自适应分区方法能够有效避免迁移分区数据传输量较大这一缺陷,进而获得了较好的数据倾斜问题处理效果.

6　结束语

针对键分布不平衡导致的数据倾斜问题,本文提出了一种自适应的Spark数据均衡分区方法,并设计实现了分区器ADBP. 该方法通过奖惩函数来调节每个分区的分配概率:每次分配后,根据各分区状态动态调整其在下一次分配中的概率,从而为每个键自适应地选择最合适的分区. 此外,对于超过分区剩余容量的键,该方法会自动根据剩余容量将键的数据划分为多个部分,并在分配概率的基础上将各个部分分配到合适的分区中. 实验结果表明:在键分布不平衡的情况下,这种自适应数据均衡分区方法能够有效缓解键倾斜现象及其带来的不利影响. 无论数据倾斜程度如何,该方法都能相对高效地均衡分区,显著提高了Spark的性能和资源利用率. 在未来的工作中,我们将对异构集群和在

线环境中的数据倾斜场景[42]进行更深入地研究,以提出更全面、更完善的Spark数据均衡分区方法.

参考文献

[1] DITTRICH J, QUIANÉ-RUIZ J A. Efficient big data processing in Hadoop MapReduce[J]. Proceedings of the VLDB Endowment, 2012, 5(12): 2014-2015.

[2] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster computing with working sets[C]//Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (Hot Cloud). Berkeley: USENIX Association, 2010: 1-7.

[3] CARBONE P, KATSIFODIMOS A, EWEN S, et al. Apache Flink: Stream and batch processing in a single engine[J]. The Bulletin of the Technical Committee on Data Engineering, 2015, 38(4): 28-38.

[4] HAN Z J, ZHANG Y J. Spark: A big data processing platform based on memory computing[C]//Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Programming. Piscataway: IEEE, 2016: 172-176.

[5] VERMA A, MANSURI A H, JAIN N. Big data management processing with Hadoop MapReduce and spark technology: A comparison[C]//2016 Symposium on Colossal Data Analysis and Networking. Piscataway: IEEE, 2016: 1-4.

[6] SHREE R, CHOUDHURY T, GUPTA S C, et al. KAFKA: The modern platform for data management and analysis in big data domain[C]//Proceedings of the 2nd International Conference on Telecommunication and Networks. Piscataway: IEEE, 2018: 1-5.

[7] ZHAI M Y, SONG A B, QIU J Y, et al. Query optimization approach with shuffle intermediate cache layer for spark SQL[C]//2019 IEEE 38th International Performance Computing and Communications Conference. Piscataway: IEEE, 2020: 1-6.

[8] YENDURI L K. Performance evaluation of apache hadoop, spark, and flink for batch processing of big data: A comparative analysis[C]//Proceedings of the 3rd International Conference on Electrical, Electronics, Information and Communication Technologies. Piscataway: IEEE, 2024: 1-5.

[9] AHN H, KIM H, YOU W. Performance study of spark on YARN cluster using HiBench[C]//2018 IEEE International Conference on Consumer Electronics - Asia. Piscataway: IEEE, 2018: 206-212.

[10] 张占峰, 王文礼, 耿珊珊, 等. Spark数据倾斜问题研究[J]. 河北省科学院学报, 2020, 37(1): 1-7.
ZHANG Z F, WANG W L, GENG S S, et al. Research on data skew of spark[J]. Journal of the Hebei Academy of Sciences, 2020, 37(1): 1-7. (in Chinese)

[11] HE Z Y, LI Z F, PENG X S, et al. DS^2: Handling data skew using data stealings over high-speed networks[C]//2021 IEEE 37th International Conference on Data Engineering. Piscataway: IEEE, 2021: 1865-1870.

[12] 何玉林, 吴东彤, Fournier-Viger Philippe, 等. 基于优先填补策略的Spark数据均衡分区方法[J]. 电子学报, 2024, 52(10): 3322-3335.
HE Y L, WU D T, FOURNIERVIGER P, et al. First filling strategy-based partitioning method to balance data in spark[J]. Acta Electronica Sinica, 2024, 52(10): 3322-3335. (in Chinese)

[13] HE Z Y, HUANG Q L, LI Z F, et al. Handling data skew for aggregation in spark SQL using task stealing[J]. International Journal of Parallel Programming, 2020, 48(6): 941-956.

[14] SHEN Y J, XIONG J, JIANG D J. SrSpark: Skew-resilient spark based on adaptive parallel processing[C]//2020 IEEE 26th International Conference on Parallel and Distributed Systems. Piscataway: IEEE, 2021: 466-475.

[15] YU J D, CHEN H P, HU F. SASM: Improving spark performance with Adaptive Skew Mitigation[C]//2015 IEEE International Conference on Progress in Informatics and Computing. Piscataway: IEEE, 2016: 102-107.

[16] HUANG Z C, WEI W G, XIE G Y. Load balancing mechanism based on linear regression partition prediction in spark[J]. Journal of Physics: Conference Series, 2020, 1575(1): 012109.

[17] 侯震梅, 杨玉莹. 分布式数据流数据倾斜均衡方法研究[J]. 长春大学学报(自然科学版), 2020, 30(5): 11-20.
HOU Z M, YANG Y Y. Research on data skew equalization method for distributed data streams[J]. Journal of Changchun University, 2020, 30(5): 11-20. (in Chinese)

[18] 周舜杰. 分布式流连接系统负载均衡策略研究[D]. 武汉: 华中科技大学, 2019.
ZHOU S J. Distributed Stream Join System Load Balance Strategy Studies[D]. Wuhan: Huazhong University of Science and Technology, 2019. (in Chinese)

[19] 卞琛, 修位蓉, 于炯. 异构Spark集群数据倾斜修正调度策略[J]. 计算机工程与科学, 2022, 44(4): 620-630.
BIAN C, XIU W R, YU J. A data skew correction scheduling strategy of heterogeneous Spark cluster[J]. Comput-

er Engineering & Science, 2022, 44(4): 620-630. (in Chinese)

[20] TANG Z, ZHANG X S, LI K L, et al. An intermediate data placement algorithm for load balancing in Spark computing environment[J]. Future Generation Computer Systems, 2018, 78: 287-301.

[21] WANG S Z, JIA Z T, WANG W L. Research on optimization of data balancing partition algorithm based on spark platform[C]//Artificial Intelligence and Security. Cham: Springer, 2021: 3-13.

[22] SONG A B, PENG B W, QIU J Y, et al. BSDP: A novel balanced spark data partitioner[C]//2021 IEEE 27th International Conference on Parallel and Distributed Systems. Piscataway: IEEE, 2022: 556-566.

[23] GUO W X, HUANG C J, TIAN W H. Handling data skew at reduce stage in Spark by ReducePartition[J]. Concurrency and Computation: Practice and Experience, 2020, 32(9): e5637.

[24] LI C L, ZHANG Y, LUO Y L. Intermediate data placement and cache replacement strategy under Spark platform[J]. Journal of Parallel and Distributed Computing, 2022, 163: 114-135.

[25] 杨迪, 赵家伟, 王鹏, 等. 面向负载均衡的动态均衡分区策略[J]. 计算机应用与软件, 2024, 41(8): 46-52.
YANG D, ZHAO J W, WANG P, et al. Dynamic balancing partition strategy for load balancing[J]. Computer Applications and Software, 2024, 41(8): 46-52. (in Chinese)

[26] KUMAR R, AGRAWAL N, TAPASWI S. Improved load balancing and partitioning model for cloud networks[C]//2024 OITS International Conference on Information Technology. Piscataway: IEEE, 2024: 803-808.

[27] SUN D W, ZHANG C L, GAO S, et al. An adaptive load balancing strategy for stateful join operator in skewed data stream environments[J]. Future Generation Computer Systems, 2024, 152: 138-151.

[28] HE C L, HUANG Y, WANG C Y, et al. Dynamic data partitioning strategy based on heterogeneous flink cluster[C]//Proceedings of the 5th International Conference on Artificial Intelligence and Big Data. Piscataway: IEEE, 2022: 355-360.

[29] IRANDOOST M A, RAHMANI A M, SETAYESHI S. A novel algorithm for handling reducer side data skew in MapReduce based on a learning automata game[J]. Information Sciences, 2019, 501: 662-679.

[30] 刘寒梅, 韩宏莹. 基于反馈调度的MapReduce负载均衡分区算法研究[J]. 信息通信, 2015, 28(10): 41-42.
LIU H M, HAN H Y. Research on MapReduce load balancing partition algorithm based on feedback scheduling[J]. Information & Communications, 2015, 28(10): 41-42. (in Chinese)

[31] YAN J T. Fuzzy-based balanced partitioning under capacity and size-tolerance constraints in distributed quantum circuits[J]. IEEE Transactions on Quantum Engineering, 2023, 4: 5100115.

[32] WANG H Y, YE X D. Research on optimization strategy of data skew problem based on hive partitioning mechanism[C]//Proceedings of the 5th International Conference on Decision Science & Management. Piscataway: IEEE, 2023: 73-77.

[33] 阎逸飞, 王智立, 邱雪松, 等. Spark环境下基于数据倾斜模型的Shuffle分区优化方案[J]. 北京邮电大学学报, 2020, 43(2): 116-121.
YAN Y F, WANG Z L, QIU X S, et al. A shuffle partition optimization scheme based on data skew model in spark[J]. Journal of Beijing University of Posts and Telecommunications, 2020, 43(2): 116-121. (in Chinese)

[34] ZVARA Z, SZABÓ P G N, LÓRÁNT B B, et al. System-aware dynamic partitioning for batch and streaming workloads[C]//Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing. New York: ACM, 2021: 1-10.

[35] 王磊. 基于Spark异构集群的低成本任务调度策略和数据倾斜优化研究[D]. 重庆: 重庆邮电大学, 2022.
WANG L. Research on Low Cost Task Scheduling Strategy and Data Skew Optimization Based on Spark Heterogeneous Cluster[D]. Chongqing: Chongqing University of Posts and Telecommunications, 2022. (in Chinese)

[36] CHEN Q, YAO J Y, XIAO Z. LIBRA: Lightweight data skew mitigation in MapReduce[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(9): 2520-2533.

[37] FU Z M, TANG Z, YANG L, et al. ImRP: A predictive partition method for data skew alleviation in spark streaming environment[J]. Parallel Computing, 2020, 100: 102699.

[38] DASH M, NG W. Efficient reservoir sampling for transactional data streams[C]//Proceedings of the 6th IEEE International Conference on Data Mining - Workshops. New York: ACM, 2006: 662-666.

[39] LIN J. The curse of zipf and limits to parallelization: A look at the stragglers problem in MapReduce[C]//Proceedings of the 7th Workshop on Large-Scale Distributed Systems for Information Retrieval (ISDS-IR). Boston:

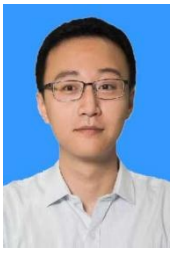
CEUR-WS, 2009: 57-62.

[40] United States Department of Transportation. Bureau of transportation statistics (BTS) [DB/OL]. (2024-04-24) [2024-04-24]. https://www.transtats.bts.gov.

[41] PONNUSAMY S, GUPTA P. Scalable data partitioning techniques for distributed data processing in cloud environments: A review[J]. IEEE Access, 2024, 12: 26735-26746.

[42] 赵二虎, 吴济文, 肖思莹, 等. 嵌入式异构智能计算系统并行多流水线设计[J]. 电子学报, 2023, 51(11): 3354-3364.
ZHAO E H, WU J W, XIAO S Y, et al. Parallel multi pipeline design of embedded heterogeneous AI computing systems[J]. Acta Electronica Sinica, 2023, 51(11): 3354-3364. (in Chinese)

作者简介



何玉林 男,1982年4月出生于河北省衡水市. 现为人工智能与数字经济广东省实验室(深圳)研究员、高级工程师. 主要研究方向为大数据系统计算技术、多样本统计分析理论与方法、数据挖掘与机器学习算法及应用.
E-mail: yulinhe@gml.ac.cn

吴东彤 女,1999年9月出生于广东省汕头市. 现为人工智能与数字经济广东省实验室(深圳)硕士研究生. 主要研究方向为大数据智能处理与分析技术.
E-mail: 2210273127@email.szu.edu.cn

黄哲学 男,1959年7月出生于黑龙江省哈尔滨市. 现为深圳大学计算机与软件学院特聘教授、博士生导师. 主要研究方向为数据挖掘、机器学习、大数据系统计算技术.
E-mail: zx.huang@szu.edu.cn