# 一种新的面向Spark跨域大数据计算的数据倾斜处理方法

何玉林[1]，常家乐[1,2]，徐贺鹏[2]，崔来中[2]，黄哲学[1]
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**

在基于Spark的跨域大数据计算中，数据倾斜会导致部分分区负载过重，从而拖慢整体任务的执行速度。现有数据倾斜处理方法多针对单数据中心设计，难以从全局层面改善跨域环境下的分区负载均衡。针对上述问题，本文提出了一种面向跨域大数据计算的数据倾斜处理方法（Skew Detection and Repartition，SDR），包括倾斜分区检测与均衡重分区两个关键操作。SDR方法首先收集各数据中心Map阶段各key的频次及对应数据量作为统计信息，并通过键值预聚合降低通信开销，采用中位数驱动的倾斜检测策略识别全局倾斜分区；随后，设计基于贪心分桶的重分区算法，将不同键值对应的数据规模作为权重，按照从大到小排序，将数据逐步分配至当前不超过目标分区大小的负载最大分区中，在保证相同键值数据不被拆分的前提下，实现各分区数据量的动态均衡，从而优化后续任务调度。实验结果表明，在不同倾斜度的数据集上，SDR方法能够有效降低分区倾斜度并缩短任务执行时间：在WordCount任务的高倾斜场景下，相比原始哈希分区，任务执行时间最高降低32.20%，在全部五个不同倾斜度的数据集上，总运行时间平均降低10.74%。

**关键词**

大数据；跨域计算；数据倾斜；重分区；负载均衡

中图分类号：TP391.9　　文献标志码：A　　doi：10.11959/j.issn.2096–0271.2025xxx

# *A Novel Data Skew Handling Method for Geo-Distributed Big Data Computing in Spark*

**He Yulin[1], Chang Jiale[1,2], Xu Hepeng[2], Cui Laizhong[2], Huang Zhexue[1]**
1. Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518107, China；
2. College of Computer Science & Software Engineering, Shenzhen University, Shenzhen 5180607, China

***Abstract***

In geo-distributed big data computing based on Apache Spark, data skew may cause excessive workloads on certain partitions, thereby delaying the overall execution progress of computing tasks. Existing data skew handling approaches are mainly designed for single data center environments and are insufficient to achieve global partition load balancing in geo-distributed scenarios. To address this issue, this paper proposes a data skew handling method for geo-distributed big data computing, termed Skew Detection and Repartition (SDR), which consists of two key operations: skewed partition detection and balanced repartitioning. Specifically, SDR first collects statistical information, including the frequency and corresponding data size of each key generated during the Map phase across multiple data centers. A key-based pre-aggregation mechanism is introduced to reduce cross-domain communication overhead during statistic collection. Then, a median-driven skew detection strategy is employed to identify globally skewed partitions. Furthermore, a greedy bin-packing-based repartitioning algorithm is designed, where the data size associated with each key is considered as its weight. The algorithm sorts keys in descending order of their weights and iteratively assigns them to the currently maximum load partition that does not exceed the target partition size. While ensuring that records with the same key are not split across different partitions, the proposed algorithm dynamically balances partition workloads and improves subsequent task scheduling efficiency. Experimental results demonstrate that SDR effectively mitigates partition skewness and reduces task execution time across datasets with different skew levels. In highly skewed WordCount workloads, SDR reduces execution time by up to 32.20% compared with the original hash partitioning method. Moreover, across five datasets with varying skew levels, SDR achieves an average reduction of 10.74% in overall execution time.

***Key words***

Big Data, geo-distributed computing, data skew, repartitioning, load balancing

第十四届CCF大数据学术会议（CCF BigData 2026）推荐稿件，编号：P00057

## 0 引言

随着互联网应用和信息技术的快速发展，各类信息系统持续产生海量数据，数据规模呈现出爆炸式增长的趋势[1]。然而，在真实应用场景中，数据通常具有明显的不均匀分布特性，不同数据之间在规模和访问频率上存在较大差异。例如，在社交网络分析、日志数据处理以及电子商务数据分析等场景中，用户行为、热点事件或热门商品往往会吸引大量关注，使相关数据在整体数据集中占据较高比例，进而导致数据分布呈现出显著的不均衡现象[2]。当利用Apache Hadoop[3]，Spark[4]等大数据处理框架对这类数据进行并行计算时，由于系统通常采用基于键值（Key）的哈希分区或范围分区策略对数据进行划分，数据的不均匀分布容易导致部分分区的数据量远大于其他分区，从而产生数据倾斜问题[5]。数据倾斜会对分布式计算系统的性能产生显著影响。在并行计算过程中，各计算节点通常需要在所有任务完成之后才能进入下一阶段，当某些节点由于处理大量倾斜数据而执行时间延长时，会导致系统整体执行效率下降[6]。此外，严重的数据倾斜还可能引发计算节点资源过载，例如内存占用过高、磁盘I/O压力增加以

及网络传输拥塞等问题，进一步降低系统的稳定性与资源利用率。

随着数据规模的扩大，现有的公司和组织会将数据保留在数据产生的地方，从而使数据呈现跨区域分布存储的特性，即计算任务所需的数据可能来自不同的数据中心[7]，这种计算任务被称为跨地理域的大数据计算。在该场景下，数据通常分散存储在多个地理位置不同的数据中心，各数据中心之间通过广域网络进行数据传输与协同计算[8]。现有多数跨域计算系统通常是在 Spark 或 Hadoop 等大数据处理框架的基础上进行扩展实现，其任务调度与执行通常以数据分区作为基本调度单元[9]。在这类框架下，数据倾斜会导致部分数据中心成为系统性能瓶颈，严重影响跨域计算任务的整体执行效率。

现有的均衡分区算法主要用于处理单集群的数据倾斜[10][11] [12] [13]。将这些算法直接应用于跨地理域场景存在明显局限性。这类方法通常仅在任务已经被分配至某一具体集群之后，才对集群内部的数据进行倾斜处理，而在跨域任务调度阶段并未显式考虑数据分布的不均衡性。若在任务分配阶段忽略数据倾斜因素，可能导致某一集群被分配远高于其他集群的数据量，从而使整体任务执行时间受限于该集群，形成显著的性能瓶颈。

针对上述问题，本文提出了一种面向跨域场景的数据倾斜检测与均衡重分区方法（Skew Detection and Repartition，SDR）。该方法通过收集各数据中心在映射（Map）阶段产生的中间键值信息，对全局数据分布特征进行分析，识别存在数据倾斜的分区，并对其进行重分区处理，从而有效提升数据分区的均衡性。经过 SDR 处理后，各分区数据规模更加接近，为后续跨数据中心的任务调度提供了更加均衡的分区输入，使得各数据中心运行的 Reduce 任务量与计算资源更加匹配，避免少数数据中心因分配过多数据量从而制约整体任务完成时间。在部署于上海、深圳、杭州三地数据中心的跨域 Spark 集群（每节点 4vCPU、16GiB 内存，数据中心间带宽 80Mbps）上，使用 WordCount 和 PageRank 两类典型大数据计算任务、每组 3GB 规模且包含五种不同倾斜程度的仿真数据集进行实验。与同样面向数据倾斜处理的 LAHP 和 SCID 算法相比，SDR 方法在不同倾斜度的数据集上均能取得更优的性能表现，验证了其在跨域计算环境下缓解数据倾斜、缩短作业执行时间的有效性。

## 1 相关工作

现有研究主要针对单集群中的数据倾斜问题，其主要分为以下静态重分区和动态负载均衡两类算法。

### 1.1 静态重分区算法

静态重分区算法需要在任务执行之前获取到数据分布信息，并根据统计结果重新设计分区策略，从而实现更加均衡的数据划分。这类方法通常通过数据采样或统计分析来估计 key 的分布情况，并据此重新划分分区边界，使得每个分区的数据规模尽可能均衡。

Xu[14]等人提出了一种基于采样的负载均衡分区方法，通过估计 key 的出现频率并将具有相同 key 的 key, value 对组织为簇（cluster）。该重分区方法首先利用 cluster 组合优化策略将较大的 cluster 分配给负载较小的规约器（reducer），以

减少reducer之间的负载差异。当数据倾斜较严重时，进一步采用cluster划分组合策略，将过大的cluster拆分为多个子cluster，并在额外的规约（Reduce）阶段合并结果，从而有效缓解数据倾斜问题。Tang[15]等人提出了一种基于采样的数据倾斜处理方法。该方法首先利用蓄水池采样估计输入数据中各键值的分布，并预测不同键对应数据集合的规模。在此基础上，将数据倾斜问题建模为装箱问题，通过对数据集合进行拆分与组合，将较大的数据集合划分后分配到不同计算节点，并采用递减首次适配策略生成负载均衡的数据分配方案。最终通过预先计算的数据放置矩阵，实现中间数据在各个归约节点之间的均衡分布。Fu[16]等人提出了一种基于历史数据预测的范围分区优化方法。该方法通过统计历史批次中各键的出现频率，并利用指数加权移动平均（Exponential Weighted Moving Average，EWMA）模型预测未来批次的键分布，在此基础上计算最优分区边界以实现负载均衡。同时，当算子语义允许时，对边界键的数据集合进行比例拆分，并结合节点处理能力在异构环境下进行数据分配。Huang[17]等人提出了一种基于异构感知的贪心优化算法。该方法首先利用改进的蓄水池采样算法在Map阶段对输入数据进行采样，并预测数据集中键值的分布情况。在此基础上，通过节点权重评估算法综合考虑节点计算能力和数据局部性，计算各节点的任务处理能力。随后，将数据倾斜问题转化为数据分配问题，采用改进的最佳适配递减算法对key簇进行分配，使其能够根据节点计算能力动态映射到不同的Reduce分区。最终实现中间数据在异构节点之间的均衡分布。Li[18]等人提出了一种基于蓄水池采样的数据放置策略。该方法首先通过随机采样预测输入数据的key分布，并构建数据倾斜度量模型以判断数据倾斜程度。当检测到严重数据倾斜时，采用细粒度数据放置策略对数据集进行切分并重新分配至多个Reduce任务；对于轻度倾斜数据，则采用粗粒度分配策略实现负载均衡。此外，该研究还提出了一种基于缓存收益最大化的自适应缓存替换算法，通过优化弹性分布式数据集（Resilient Distributed Dataset，RDD）缓存策略以减少重复计算开销，从而进一步提升Spark作业执行效率。

### 1.2 动态负载均衡算法

与静态重分区方法不同，这类研究通过在任务执行过程中动态监测系统负载情况，并根据运行时状态对任务进行重新调度或负载迁移，从而实现更加灵活的负载均衡。这类方法通常不需要提前获取完整的数据分布信息，而是通过运行时调度机制来逐步缓解数据倾斜问题。除面向数据倾斜的动态负载均衡研究外，Spark作业调度研究还关注资源使用成本、作业响应时间和服务质量达成率等优化目标。

Kwon[6]等人提出了一种基于运行时任务重调度的负载均衡方法，通过在作业运行过程中监控任务执行进度来识别拖尾任务。当检测到负载不均时，该方法会对慢任务剩余的未处理数据进行重新划分，并将其拆分为多个子任务分配给空闲节点执行，以减少节点之间的负载差异。此外，该方法设计了细粒度的数据迁移机制，可以在不中断任务执行的情况下提取未处理数据并重新调度执行，从而避免重新运行整个任务，有效缓解数据倾斜问题。He[19]等人针对Spark SQL聚合操作中的数据倾斜问题，提出了一种基于任务窃取

（Task Stealing）的动态负载均衡方法。通过在运行时将热点分区中的部分计算任务迁移至空闲执行器，从而有效缓解因倾斜导致的执行瓶颈。该方法无需改变查询语义或预先了解数据分布，保持了系统的透明性。卞琛[20]等提出了一种面向异构资源感知的数据倾斜修正调度策略。该方法在任务调度阶段综合考虑节点计算能力差异与分区数据规模，通过对倾斜任务进行自适应拆分与重分配，使计算负载在异构执行节点之间更加均衡。SrSpark[21]是针对静态任务划分下因数据倾斜导致执行时间增加的问题，提出的一种基于自适应并行处理的抗倾斜处理算法。SrSpark 在运行过程中检测任务执行状态，对倾斜任务进行在线划分。但是它主要应用于 Spark SQL 中，无法扩展到普通的计算中。Irandoost[22]等人提出了基于学习自动机的自适应分区（Learning Automata Hash Partitione，LAHP）算法。在不依赖预统计数据分布的情况下，仅根据节点负载动态调整键和节点之间的映射关系。它允许热点键在线拆分，按照 reducer 计算能力进行分配数据，实现集群的负载均衡。Yang[23]等人提出了一种基于自适应计算粒度调整的均衡分区策略。该方法首先利用优化的步长采样算法估计中间数据的 key 分布，并构建数据倾斜模型，通过最大公约数计算动态调整分区粒度。在此基础上，将 key 划分为高权重 key 和低权重 key，并分别采用基于贪心策略的高权重 key 分区算法和基于哈希与伪随机探测的低权重 key 分区算法进行数据分配，从而实现 reducer 负载均衡并减少任务执行时间。

# 2 预备知识

在本节中，介绍了基于 Spark 实现的跨域去中心化计算框架以及跨域任务调度。

## 2.1 跨域去中心化计算框架

现有的基于 Spark 的跨域大数据处理架构通常可分为集中式架构与去中心化架构两种。本文的算法应用于去中心化架构中。如图 1 所示，在去中心化架构中，系统由多个主节点和若干个工作节点构成。每个数据中心内部部署一个主节点，该主节点仅负责管理数据中心内的计算资源，并调度工作节点去执行具体的任务。当作业所需的数据均存储在单一的数据中心时，该数据中心可作为一个独立的计算集群，在本地完成任务调度与执行，无需跨域协同。当作业涉及的数据分布在多个数据中心时，多个数据中心将动态组成跨域集群组。作业被提交至跨域集群组，提交该作业的数据中心主节点将作为全局主节点，负责跨数据中心层面的任务调度与全局资源协调。全局主节点首先根据跨域作业构建有向无环图（Directed Acyclic Graph，DAG），之后采用与 Spark 一致的阶段划分规则，根据分区之间的宽依赖，将作业分为多个任务阶段。每个任务阶段都需要进行任务调度。按照 DAG 顺序执行任务阶段[24]。

全局主节点在数据中心粒度上进行任务划分与分配，将子任务分派至参与的各个数据中心；随后，数据中心内部的主节点再依据本地计算框架（如 Spark）的调度机制，对任务进行细粒度的资源分配与执行管理。

## 2.2 跨域任务调度

定义一个跨域计算系统，由 $N$ 个数据中心组成，各数据中心通过分布式文件系

统（Hadoop Distributed File System，HDFS）存储数据。在该文件系统中，数据以数据块的形式进行管理，默认块大小为128MB。在基于Spark的计算框架中，任务数是由数据分区数量决定的，即每个分区对应一个计算任务。在默认配置下，数据分区的划分方式与HDFS数据块保持一致，因此单个分区的数据量通常与数据块大小相当。

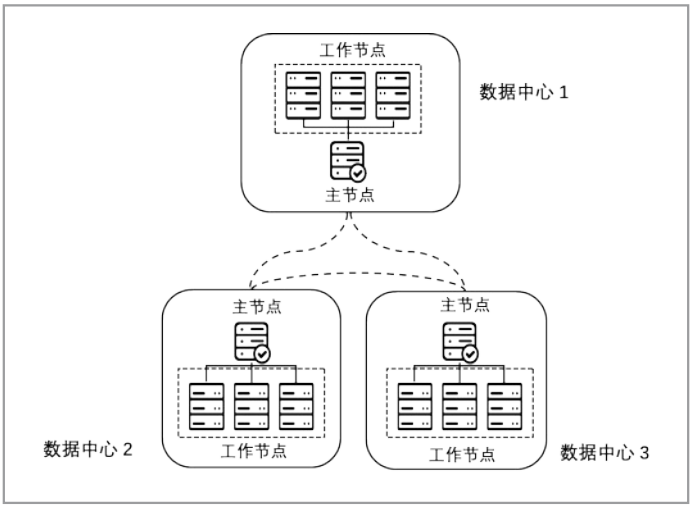



图1 去中心化架构

记跨域作业所需的总数据量为$S$，其分布式存储在不同的数据中心。第$i$（$i\in\{1,2,\cdots,N\}$）个数据中心的数据量为$S_i$，且满足$\Sigma S_i=S$。因为数据中心内部的计算框架与Spark类似，所以数据划分的粒度定义为$D_{def}$=128MB，并据此完成原始数据到数据块的转换。之后，数据块被直接映射为调度单元，即每一个数据块对应一次独立的任务执行。因此，数据中心$i$中包含的数据块数量$B_i$即表示其初始可调度任务规模，其计算方式如下所示：

$$B_i=\frac{S_i}{D_{def}} \tag{1}$$

在后续调度过程中，不再区分数据量与任务数两个概念，而是统一以数据块数量进行描述，即任务分配的结果可以直接表示为各数据中心所获得的数据块数。对于计算能力，本文采用资源规模进行抽象描述。具体地，将数据中心$i$的计算能力表示为$E_i$，其数值等于该数据中心内计算节点数量与单节点CPU核心数的乘积。该指标用于衡量数据中心在同一时间内可并发执行的任务上限。

在该系统中，采用点对点的网络架构，数据中心$i$和数据中心$j$（$j\in\{1,2,\cdots,N\}$）之间通过点对点链路直接连接。数据中心之间可以直接进行数据传输，不需要考虑中心网络拥塞问题。

在基于Spark的计算框架中，任务通常被划分为两个主要阶段：Map阶段和Reduce阶段。二者的主要区别在于，Map阶段仅涉及数据的转换类操作，通常可以在数据所在的数据中心内部完成，无需进行跨数据中心的数据传输。

跨域环境下的Map阶段仍然遵循数据本地性原则，即各数据中心优先对本地数据块执行计算任务，从而避免不必要的数据迁移。因此，跨域Map阶段本质上可以视为多个数据中心内部并行执行的本地计算过程，各数据中心之间基本不存在数据交互，其性能主要受限于各自的计算资源。

相比之下，Reduce阶段通常涉及基于key的数据聚合操作，这些操作涉及跨域混洗（Shuffle）操作，需要将具有相同key的数据从不同数据中心重新分布到同一目标节点。在跨域场景中，这一过程不可避免地引入跨数据中心的数据传输，即跨域通信开销。具体来说，在Shuffle过程中，Map阶段生成的中间结果需要按照key进行分区，并通过网络传输到对应的Reduce任务所在的数据中心，从而完成后续的聚合计算。因此，Reduce阶段是影响系统性能的关键瓶颈，所以跨域任务调度主要集中于Reduce阶段。

整个跨域任务调度的流程如图2所示，每个数据中心的数据都被划分为相同数量的分区，其数量 $R_{total}$ 可由总数据量 $S$ 以及单个数据分片大小 $D_{def}$ 计算得出：

$$R_{total}=\frac{S}{D_{def}} \tag{2}$$

跨域任务调度的核心在于确定各分区任务在不同数据中心之间的分配策略。具体来说，每个分区需要被分配至某一目标数据中心执行对应Reduce任务。当某一分区被分配至数据中心 $i$ 时，其余数据中心中属于该分区的数据需要通过网络传输至数据中心　，从而完成该分区数据的汇聚。在所有相关数据传输完成后，目标数据中心即可对该分区执行Reduce计算任务。

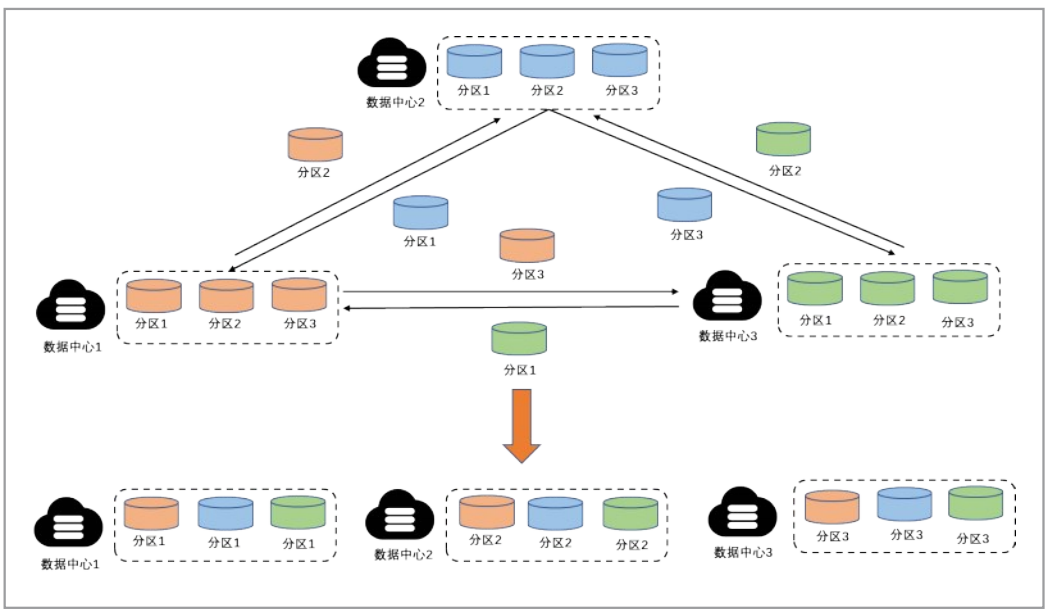



图2　跨域任务调度流程

## 3 SDR数据倾斜检测与重分区方法

在基于分区的跨域任务调度中，通常假设各分区的数据规模相对均衡，从而认为每个分区对应任务的数据传输时间与计算时间基本一致。然而，实际数据往往呈现非均匀分布特征，导致分区之间的数据量存在差异。若仍按等规模分区进行调度，可能使部分数据中心承担与其计算资源不匹配的任务负载，进而导致实际执行时间与调度预期产生较大偏差。

针对上述问题，本文提出了一种面向基于Spark的跨域计算均衡重分区算法。该算法首先基于中位数对全局分区进行数据倾斜检测，然后针对倾斜分区采用单key不拆分的贪心分桶策略进行重分区，从而提升分区均衡性，保证后续任务调度的有效性。

### 3.1 数据倾斜检测算法

图3展示了SDR算法的整体流程，主要分为数据倾斜检测和重分区两部分。对于数据检测部分，首先需要解决的核心问题是数据收集阶段的通信开销。若直接将所有中间数据集中传输至单一数据中心进行数据倾斜检测，不仅会产生大量跨数据中心的数据传输开销，还会显著增加时间成本与经济成本，甚至可能抵消后续调度优化所带来的性能收益。因此，本文在设计中避免对完整数据进行集中式收集，而是利用跨域Shuffle阶段的数据特性进行优化。

在跨域Shuffle过程中，中间数据通常以键值对 key, value 的形式存在，并在基于哈希分区策略下根据key的哈希值进行分区。相较于value，key具有体量小、重复率高等特点，而数据倾斜问题本质上源于少数高频key的不均匀分布。基于此，本文仅收集各数据中心中间数据中的key统计信息（key的频率），用于全局数据分布分析，从而在保证分析有效性的同时显著降低跨域数据传输规模。

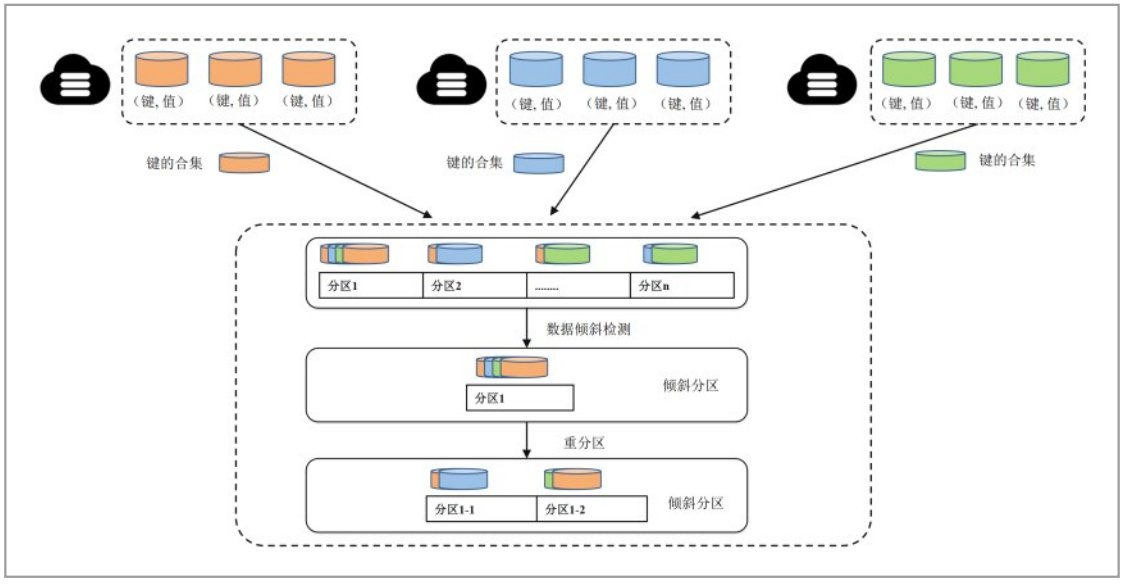



图3 SDR算法流程

进一步地，为减少key值数据在跨域传输过程中的通信开销，并提升后续倾斜检测的效率，本文在数据中心本地引入了key的预聚合操作。具体而言，各数据中心在发送数据之前，先对本地中间数据中的key进行聚合，将相同的key合并为一条记录，生成形如 key, size 的统计信息。通过该预处理步骤，可以压缩需要跨域传输的数据规模，一方面减少公网传输所需的时间和带宽消耗，另一方面降低后续全局数据倾斜检测阶段的计算复杂度。在完成key统计信息的收集后，系统即可基于各分区的key分布情况识别潜在的数据倾斜分区。

本文采用基于中位数的数据倾斜检测算法。算法的伪代码如算法1所示，首先，根据上一个任务阶段预聚合后的各个数据中心的 key, size 数据，将其按照key进行哈希分区，统计每一个分区的数据量（第1-7行）。之后，对所有分区的数据量进行排序并计算其中位数（第8-13行）。在此基础上，算法设置倾斜检测阈值为中位数的固定倍数$\beta$，$\gamma$（第15行），并逐一遍历所有分区（第17行），将数据量超过该阈值或数据量小于阈值的分区标记为倾斜分区（第18-20行）。如果没有检测到倾斜分区，则最终的倾斜分区集合为空集。

算法将$\beta$设置为2，$\gamma$设置为0.2，即当分区内数据量大于分区数据量2倍或者小于分区数据量0.2倍时会被判定为倾斜分区。$\beta$默认为2，$\gamma$默认为0.2是因为默认情况下分区数是由数据量的大小决定。在无数据倾斜的理想情况下，每个分区的数据应该都是$D_{def}$，分区中位数稳定在128MB。当出现数据倾斜时，只有少部分分区的数据量显著大于$D_{def}$，或者数据量显著小于$D_{def}$，而大部分分区数据量依旧能够保持在默认分区大小附近。由于倾斜分区数量$|Q|$通常远小于分区总数$R_{total}$，即$|Q| \ll R_{total}$，因此分区数据量分布的中位数仍然由非倾斜分区决定，从而在$D_{def}$附近波动。在任务执行阶段，Spark中每个分区对应一个并行任务，其执行时间与分区数据量近似成正比。对于倾斜分区，若将其重新分区，分区数据更加均衡，总体计算时间会减少，但同时会引入额外的重分区开销，所以只有当倾斜分区数据量显著大于其他分区或者数据量明显小于其他分区时，拆分的收益才能抵消这些额外的开销，这样重分区才有实际意义。

**算法1 跨域数据倾斜检测算法。**

输入：数据中心预聚合的数据$Key(key,size)$，分区数$R_{total}$，倾斜判定系数$\beta,\gamma$；
输出：倾斜分区集合$Q$。

```
1: for i in R_total do:
2:     d_i=0;                          /*初始化第i个分区的数据量*/;
3: end;
4:for each (key,size) in Key do:       /* 根据hash规则确定所属的分区*/
5:     i= hash(key) mod R_total;
6:     d_i = d_i + size;
7: end;
8:Sort(d_1,d_2,…,d_Rtotal);            /*按分区数据量升序排序*/
9: if R_total is odd then:
10:    Median = d_(total+1)/2;
11: else:
12:    Median = (d_Rtotal/2+d_R(total/2+1))/2;
13:end;
14:Q =∅;
15:Threshold1 = β * Median, Threshold2 =γ*Median; /*确定阈值*/
16: for i in R_total do:
17:    if d_i > Threshold1 || d_i < Threshold2 then:
                                       /*根据阈值判定倾斜分区*/
18:        Q ∪ {i};
19:    end;
20: end;
21 return Q;
```

### 3.2 平衡重分区算法

在检测到数据倾斜分区之后，需要对倾斜分区执行重分区操作以实现负载均衡。在单数据中心场景下，重分区过程允许将同一key的数据分配到多个分区中，从而通过拆分分区缓解由于单key数据量过大而导致的计算瓶颈。然而，该策略会破坏后续计算阶段对key一致性的约束，即同一key的数据需要由同一计算节点统一处理。为保证计算结果的正确性，在各计算节点完成局部计算后，需要引入额外的聚合阶段，对被拆分的同一key的中间结果进行重新合并。此外，由于再聚合阶段传输的数据通常为中间结果，其规模显著小于原始输入数据，且单数据中心内的计算节点通过高速、低延迟的局域网互联，使得额外引入的网络传输与计算开销相对有限。从整体调度角度来看，单key拆分能够有效提升任务并行度，因此在单数据中心环境下是一种行之有效的数据倾斜缓解手段。

然而，在多数据中心场景中，单key拆分的策略具有一定的局限性。与单数据中心内部依赖高速、低延迟局域网不同，不同数据中心之间通常通过广域网进行互联，其带宽受限。当同一key的数据被拆分并分布至多个数据中心后，后续计算阶段需要执行跨数据中心的再聚合操作。该过程会增加大量的跨域数据传输，放大广域网对整体任务执行时间的影响，增加额外的网络开销。而且，频繁的跨域聚合还可能成为整体任务执行的瓶颈。因此，单key跨分区拆分的策略不适合跨域计算场景。所以，本文的重分区算法对于单个key不进行拆分，使用贪心策略重新分配数据。

具体的重分区算法伪代码如算法2所示。由检测算法可知倾斜分区集合$Q$，那么重分区算法只需要对倾斜分区数据操作即可。首先，需要遍历倾斜分区中的所有数据 key, size ，将相同*key*的数据量累加，得到每个key对应的总数据量*Size(key)*（第1-3行）。在完成key聚合之后，算法按照*Size(key)*对所有key进行降序排列（第4行），这样可以使得数据量较大的key优先参与分桶。之后采用贪心策略进行重分区，初始化桶集合$B$，其中每个桶对应未来一个新的分区（第5行）。之后对降序排序后的key进行逐个处理，对于当前key，先在已经创建的桶集合中查找一个桶$b$，保证该桶加入key后桶的负载不超过$C_{target}$目标容量。同时还需要满足在满足约束的桶中，*Size(b)*是最大的，使桶的负载更加接近$C_{target}$（第7行）。如果存在满足条件的桶，就将key分配到该桶中，更新桶的负载。如果不存在，则创建新的桶*b_new*，并将该key单独分配至新桶中（第8-14行）。当所有key完成分桶之后，如果存在一个桶它的数据量小于$0.2*C_{target}$，即生成了一个小分区，那么将这个桶的key依次分配至其他桶，优先分给最小的桶。算法为每个桶分配唯一的新分区编号*pid*，并构建key到新分区的映射关系。具体而言，将桶中所有key映射至该桶对应的分区*pid*，形成映射表$Key_{re}$。

在重分区算法中$C_{target}$默认设置为当前作业中分区数据量的中位数。之所以选择中位数作为重分区的目标容量，相比均值，中位数对极端大分区不敏感，能够更加真实地反映大多数分区的实际数据规模。举例设存在$v$个倾斜分区，其数据量远大于正常分区：

$$d_j \gg d_{Median}, j \in Q, |Q| = v, v \ll R_{total} \quad (3)$$

其中$d_j$是倾斜分区的数据量，$d_{Median}$是分区中位数对应的数据量。分区的均值为：

$$\mu = \frac{1}{R_{\text{total}}} \sum_{i=1}^{R_{\text{total}}} d_i \tag{4}$$

可得：

$$\mu = \frac{1}{R_{\text{total}}} \left( \sum_{i \notin Q} d_i + \sum_{j \in Q} dj \right) \tag{5}$$

由于 $d_j \gg d_{Median}$，即使 $v \ll R_{total}$，也有 $\mu \gg d_{Median}$。可以看出均值在某些情况下会丢失代表性。而且，在以最小化分区负载的绝对偏差为目标时，最优目标值正是分区数据量的中位数。因此，这种操作使重分区后的新分区在数据规模上尽可能与非倾斜分区保持一致，避免出现重分区后仍显著偏大或偏小的情况，使得后续任务执行阶段中各并行任务的负载更加均衡。

在上述重分区映射关系构建完成之后，将进入到数据中心重分区阶段。重分区算法仅针对倾斜分区集合 $Q$ 进行操作，非倾斜分区的数据保持不变，所以能够减少不必要的数据移动。各数据中心在接收到映射表之后，仅需要对本地倾斜分区中的数据按照映射关系重新分配至对应的新分区，从而完成重分区的过程。在这个过程中不涉及跨数据中心的数据传输，避免了在重分区阶段额外地引入广域网的通信开销。

**算法2　跨域平衡重分区算法。**

输入：倾斜分区集合 $Q$，目标分区容量 $C_{target}$，倾斜分区数据 $Key_Q(key, size)$；
输出：重分区后的数据 $Key_{re}$。

```
for each (key,size) in Key_Q do:        /*统计每个key对应的数据量*/
    Size(key) = Size(k) + size;
end;
Sort(key by Size(key));
bucket set B = {};
for (key,size) in Key_Q do:
    找到一个桶 b∈B，满足 Size(b)=Max(B) ，Size(b) + Size(key)<C_target
    if b exists then:
        将key分配至桶b;
        Size(b) = Size(b) + Size(key);
    else:
        创建新桶 b_new，将key分配至桶 b_new;
        Size(b_new) = Size(b_new) + Size(key);
end;
for b in B do:                          /*分配新分区*/
    为每一个桶分配一个分区 pid；
    for key k in b do:
        Key_re(k) = pid;
    end;
end;
return Key_re;
```

## 3.3 算法复杂度分析

在本节中，将对倾斜检测以及重分区算法的时间复杂度和空间复杂度进行详细的分析。

### 3.3.1　时间算法复杂度分析

倾斜分区检测算法：假设有 $N$ 条 key, size 的记录，需要对这 $N$ 条数据进行遍历，并通过哈希函数将其映射到对应分区，同时累加分区数据量。哈希操作以及累加操作的时间复杂度均为常数，所以该阶段的时间复杂度为 $O(N)$。之后需要对分区进行排序，并求取中位数，分区数量为 $R_{total}$，所以其时间复杂度为 $O(R_{total} \times \log R_{total})$。后续的遍历所有分区比较得到倾斜分区，时间复杂度为 $O(R_{total})$。又因为在大数据场景中，有 $R_{total} \ll N$，即分区的数量要远小于 key, size 的数据量。所以可以得出倾斜分区检测算法最终的时间复杂度为 $O(N)$。

平衡重分区算法：假设倾斜分区中有 $M$ 条不同 key 的数据，重分区后的分区数量为 $B_{re}$。算法遍历倾斜分区集合，在同一

分区内，将同一key的数据量进行累加操作，该操作的时间复杂度为$O(M)$。之后对key按照其数据量降序排列，排序操作的时间复杂度为$O(M\times\log M)$。排序操作结束后需要进行重分区，每一个key需要找到合适的分区放入，最坏的情况就是需要将现有的所有都遍历一遍，然后无法满足，创建一个新分区。所以，其时间复杂度为$O(M\times B_{re})$。综上，算法的整体时间复杂度应为$O(MlogM+M\bullet B_{re})$。

#### 3.3.2 空间算法复杂度分析

倾斜分区检测算法：算法需要维护一个长度为$R_{total}$的数组，用于存储每个分区的数据量，所以其空间复杂度为$O(R_{total})$。此外，在排序过程中也需要额外的空间，在最坏情况下空间复杂度为$O(R_{total})$。所以，可以得到算法整体的空间复杂度为$O(R_{total})$。

平衡重分区算法：计算key的数据量这一操作，需要$O(M)$空间用于存储每个key的累计数据量。之后需要创建$B_{re}$个新分区以及分区中的所有key数据，所以需要$O(M+B_{re})$的空间。最后，在将key映射到分区的步骤中，需要为每个key存储一个分区编号，其空间复杂度为$O(M)$。综上，该算法的空间复杂度为$O(M+B_{re})$。

## 4 实验与评估

本节将本文提出的SDR算法与单集群中处理数据倾斜的算法LAHP和SCID算法进行对比。通过在多数据中心环境、数据倾斜的情况下分别执行两类具有代表性的大数据计算任务WordCount与PageRank，以评估所提算法的有效性。

### 4.1 实验设置

本节实验采用阿里云的EMR on ECS服务，在上海、深圳、杭州等地部署多个地理分布式独立计算集群，模拟跨地域大数据协同计算场景。每个集群内部均部署Spark 2.4.0作为大数据处理引擎，并以YARN作为资源与任务调度管理组件。每个集群的1个工作节点运行Spark中执行任务的1个Executor。各集群的工作节点均为相同配置，以保证计算资源的一致性，每台机器均配置为4vCPU、16GiB内存，集群内部网络带宽为1Gbps。在实验中，每个任务阶段总的分区数量即任务总数由每个阶段的输入数据总量$S$以及每个分区的默认大小$D_{def}$决定，分区数量为$S/D_{def}$，其中$D_{def}$设为128MB。

实验选取了两个常见的大数据任务，词频统计（WordCount）和网页排名（PageRank）。由于需要不同倾斜程度的数据集进行实验以验证本文算法的有效性，现有的真实数据集数据量较小且倾斜程度无法控制，所以为两类任务均构建了仿真数据集。每种任务类型包含了5个不同倾斜度的数据集。每个数据集进一步划分为三个子数据集。每一个子数据集的大小都为3GB，分布在三个数据中心中。

现有的数据倾斜处理算法都是单数据中心的，所以对比实验中选取的算法主要面向单数据中心场景，并不会改变全局分区的倾斜度。在跨域场景下，其具体实验流程为：当任务调度至各数据中心后，在正式执行计算前，在中心内部运行该倾斜处理算法，对本地待处理数据进行重分布和优化。所以，在与LAHP和SCID算法进行对比实验时，其任务调度还是默认为全局分区是均衡的去分配任务。

### 4.2 评价指标

为了能够量化分区的数据倾斜的程度，本节引入变异系数（Coefficient of Variation, CV）作为衡量指标。变异系数通过刻画数据分布相对离散程度来反映倾斜强度。变异系数的定义为：

$$CV=\frac{\sigma}{\mu} \tag{6}$$

其中，$\sigma$表示各分区数据量的标准差，$\mu$表示分区数据量的均值。变异系数越大，说明分区之间的数据规模差异越显著，即数据倾斜越严重；当CV接近0时，表明所有分区的数据规模较为均衡，数据倾斜现象较弱或者不存在。

数据倾斜最直接的影响表现为下一个任务阶段执行时间的延长。当部分分区数据量远大于其他分区时，对应的计算节点需要处理更多的数据，从而拖慢整个阶段的完成时间。因此，倾斜检测和重分区操作的目标，是使分区分布更为均衡，使不同数据中心获得的有效任务量更加接近，从而缩短后续阶段的处理时间。

基于此，实验中设置了两个主要性能指标：

**1. 下一个任务阶段的执行时间**，该指标用于衡量分区完成后实际任务执行的效率，包含任务的计算开销以及跨域数据传输开销。分区越均衡，任务调度越合理，执行时间理论上越短。

**2. 下一个任务阶段的总运行时间**，在执行时间的基础上加入了重分区过程本身所消耗的时间，反映算法在真实运行环境中的整体代价。由于本文算法在获取重分区映射表后，各数据中心需要同步完成分区调整，而SCID算法在生成新的分区方案时也需要额外消耗时间，因此总运行时间能够更客观地反映算法在实际部署场景下的整体效果。

执行时间与总运行时间越小，说明重分区算法在缓解数据倾斜、提高跨域任务效率方面表现越好。

### 4.3 算法可行性验证

本小节实验旨在验证所提出SDR算法在数据倾斜处理方面的可行性。为此，将SDR算法与原始哈希分区算法以及LAHP和SCID算法进行对比。实验选取了五组具有不同倾斜程度的WordCount数据集，并采用仅包含一个Reduce阶段的WordCount程序作为测试任务。一共选取3个数据中心，每个数据中心具有3个计算节点，数据中心之间的网络带宽均为80Mbps。

首先，对比各算法在重分区前后全局数据分区倾斜度的变化情况。需要说明的是，LAHP和SCID均为面向单数据中心场景设计的倾斜处理算法，其主要处理数据中心内部的数据倾斜，并不会对全局分区结构进行调整。因此，在这两种算法中，执行倾斜处理前后的全局分区倾斜度保持不变。基于这一特性，在全局分区倾斜度的对比分析中，不再对LAHP和SCID的重分区前后结果进行比较。

由图4可以看出，与原始哈希分区相比，执行SDR算法后各数据集的分区倾斜度均得到降低。随着数据集1至数据集5的数据倾斜程度逐步加大，原始分区的倾斜度持续上升，而SDR算法在不同倾斜度下均能保持稳定的优化效果，其重分区后的倾斜度增长幅度受限。实验结果表明，SDR算法对不同程度的数据倾斜具有良好的适应性，能够改善分区负载不均问题，为后续跨域任务调度提供更加均衡的数据分区。

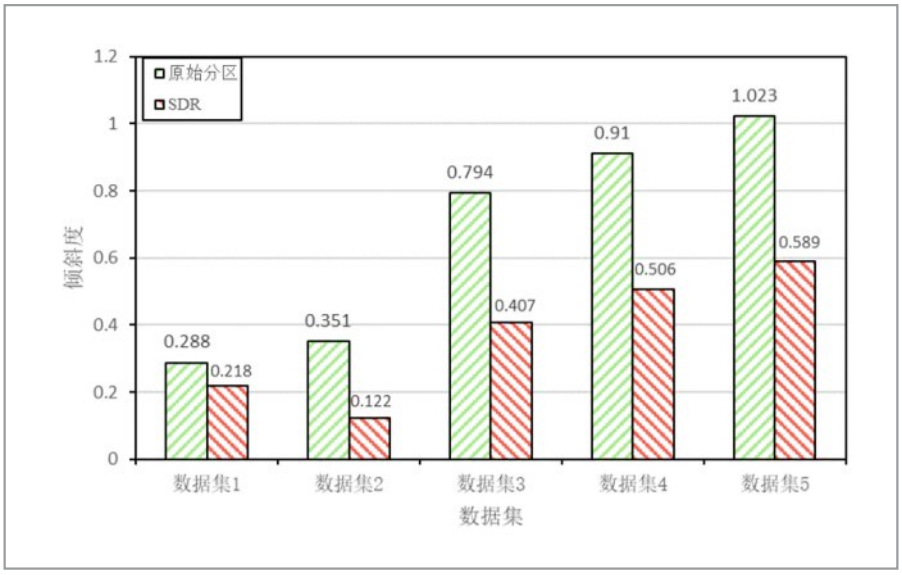



图4 重分区前后倾斜度对比

图5是WordCount任务中Reduce阶段的执行时间，其中包括了数据传输时间以及任务计算时间。可以看出，随着数据集倾斜度的上升，任务阶段的执行时间也随之上升。但是，相较于LAHP和SCID算法，SDR算法执行时间的增长幅度较小。其中，在数据集1下也就是倾斜度最小时，SDR相比原始分区方法执行时间下降8.43%。而在数据集5也就是倾斜度最大时，执行时间下降32.20%，可见随着倾斜度的提升，SDR数据倾斜处理的效果更显著。虽然LAHP和SCID对于数据倾斜的处理也有一定效果，在数据集5中，LAHP和SCID相较于原始分区分别降低约5.86%和9.01%的执行时间，但是下降幅度有限。这是因为这两个算法都只能在数据中心内部执行任务时进行分区均衡，无法提前检测到总体数据的数据倾斜，即只能降低计算时间，但是无法降低跨域数据传输时间。对于WordCount任务的Reduce阶段，它的任务执行时间中占比较大的是跨域数据传输时间，所以LAHP和SCID的重分区效果有限。而SDR算法是对全局分区进行重分区，并且将重分区信息传递给后续的任务调度，可以综合考虑选取执行时间最低的任务方案进行任务分配。

图6是WordCount任务中Reduce阶段的总运行时间，包括了执行时间和分区时间。因为SDR算法根据key值数据对全局分区进行重分区之后，需要将分区映射表发送到每一个数据中心，数据中心会根据分区映射表对本地数据中的倾斜分区进行重分区，这个重分区操作是需要耗费时间的。此外，SCID算法也需要进行获取分区方案的操作，这个也是需要额外消耗时间。所以，总运行时间能够更能反映实际场景下数据倾斜处理算法的效果。可以看出，相比图5只有SCID与SDR时间有所增加。在低倾斜度数据集，SDR效果一般，其效果只是次优，但是随着倾斜度增加，在数据集5效果达到最优，其降低时间达到20.72%。五个数据集平均降低时间10.74%，证明了SDR算法在不同倾斜度下均具有可行性。

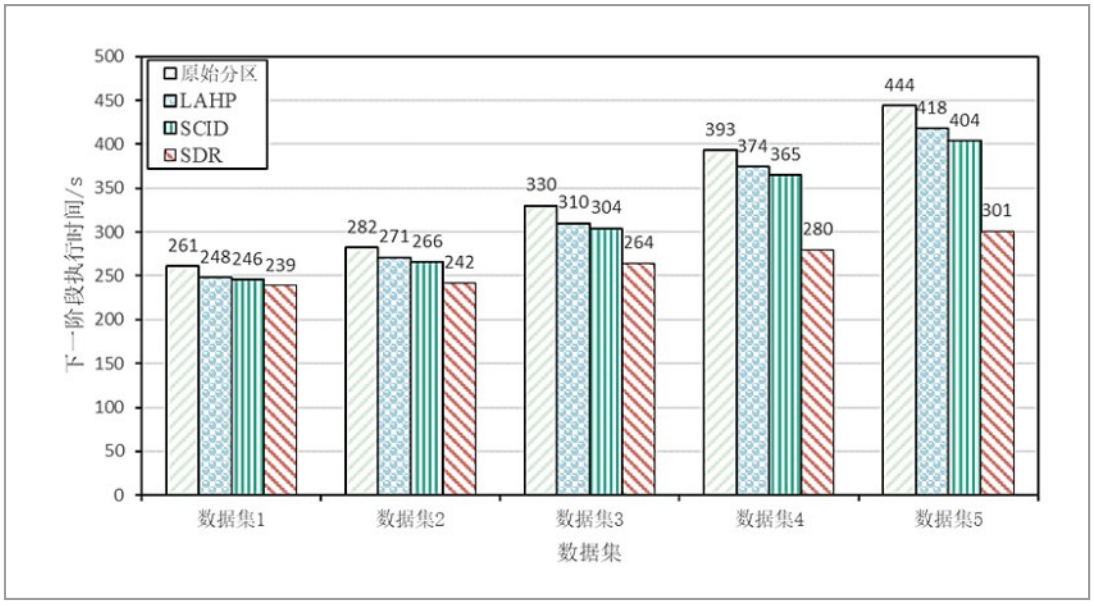



图5 WordCount中下一个任务阶段的执行时间

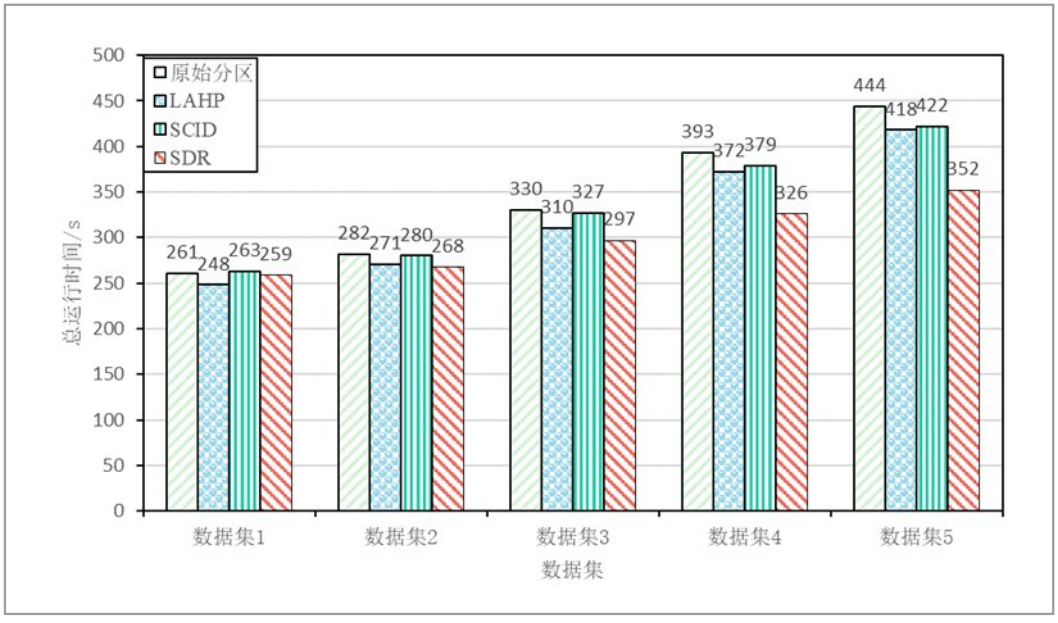



图6 WordCount中下一个任务阶段的总运行时间

## 4.4 算法有效性验证

本小节实验是验证所提出SDR算法在数据倾斜处理方面的有效性。基于Spark的计算任务绝大部分为多任务阶段，即存在多个Reduce任务阶段，需要进行多次Shuffle操作。为了能够更加真实地验证算法在实际生产环境中的效果，本节实验的跨域任务采用PageRank，PageRank包含多个任务阶段，而且是具有代表性的大数据程序。实验选取了五组具有不同倾斜程

度的数据集，分成三个子集分布在3个数据中心，每个数据中心具有3个计算节点，数据中心之间的网络带宽均为80Mbps。

图7展示了PageRank第一个Reduce任务阶段的执行时间及对应的总运行时间。由于该阶段的核心操作为distinct，在该操作中，每条记录本身作为key。实验所使用的数据集在该阶段不存在重复记录，因此各分区接收到的数据量基本一致，分区数据分布呈现完全均匀状态，不同数据集对应的分区倾斜度均为0，故未单独展示其倾斜度结果

在分区均匀的前提下，该阶段无需进行额外的重分区或倾斜处理操作，因此不存在额外的重分区开销，任务阶段的执行时间与任务的总运行时间基本一致。同时，由于不存在数据倾斜问题，各种数据倾斜处理算法在该阶段的执行效果相同，表现出的运行时间差异可以忽略不计。

需要指出的是，即使在分区均匀的情况下，不同数据集在该阶段的执行时间仍存在一定差异。这是因为各数据集在Map阶段后的数据规模并不完全一致，数据量的变化与原始数据分布特征相关，从而导致后续Reduce阶段的处理数据规模不同，最终造成了执行时间上的差异。

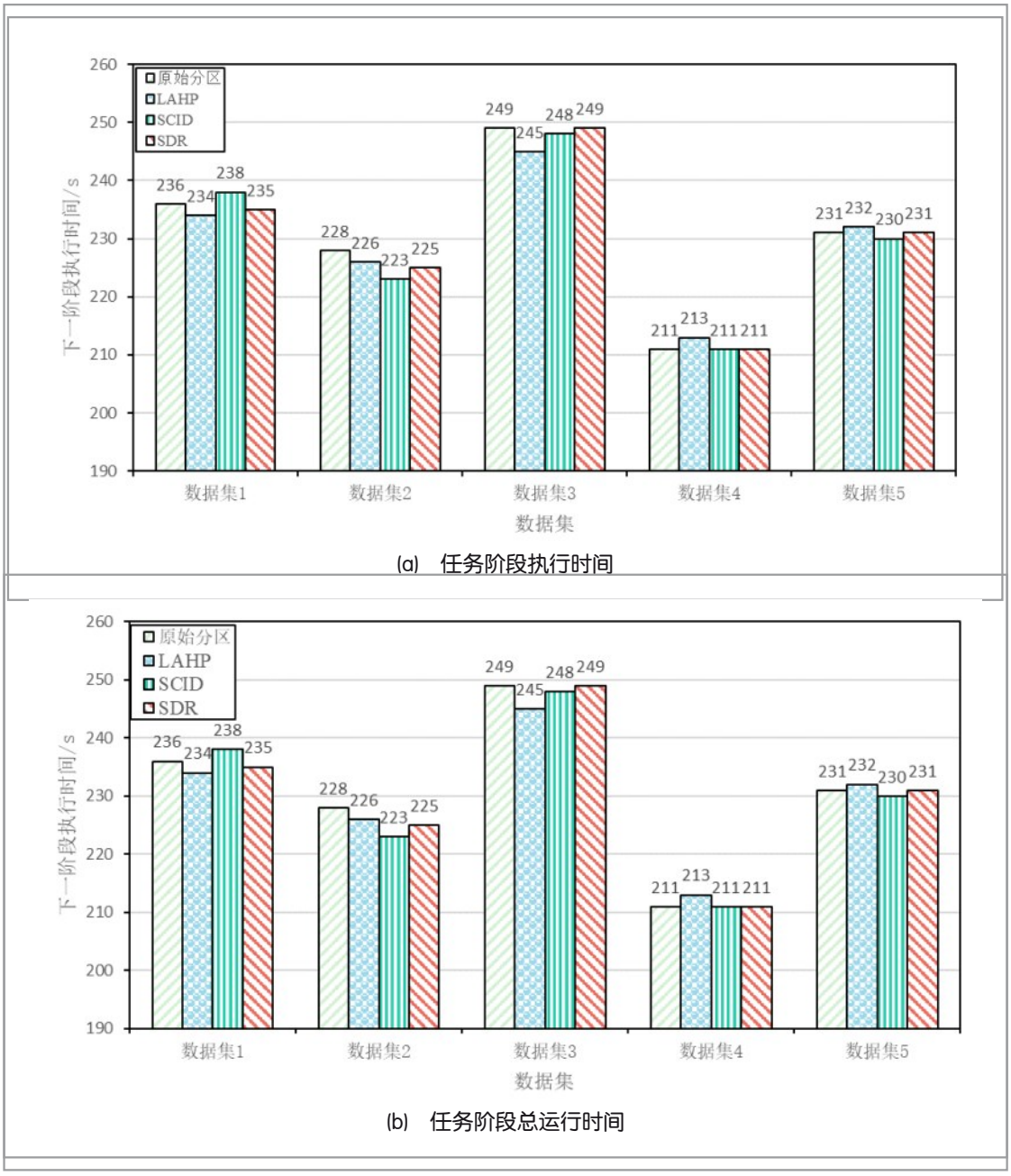
图8展示了PageRank第二个Reduce任务阶段在不同数据集下的执行与总运行时间，而图9给出了该阶段对应的总数据分区倾斜度。由图9可以看出，SDR算法能够有效抑制倾斜分区的产生，使各分区的数据规模保持较为均衡。进一步结合图8可以发现，相较于LAHP和SCID算法，SDR算法在该任务阶段表现出更低的执行与总运行时间。

整体数据倾斜度较低时，SDR算法与LAHP算法的性能表现较为接近，对应数据集1至4。在数据倾斜最为严重的数据集5中，SDR算法在图8(b)所示的任务总运行时间方面明显优于LAHP算法。LAHP算法相较于原始分区方式可将总运行时间降低3.0%，而SDR算法可降低8.43%。在全部5个数据集上，SDR算法相较于原始分区方式平均降低总运行时间6.6%。

图10展示了第三个Reduce任务阶段在不同数据集下的执行时间与总运行时间，图11给出了该任务阶段在SDR算法重分区前后总体数据分区倾斜度的对比结果。由图11可以看出，数据集1和数据集3的整体倾斜度较低，未触发重分区操作。然而，从图10中可以观察到，即使在未进行重分区的情况下，SDR算法相较于原始分区方案，其任务执行时间和总运行时间仍然有所降低。

其原因在于，前一任务阶段中原始分区策略未对数据倾斜进行有效控制，导致在进入第三个任务阶段时，不同数据中心之间的数据量分布已经出现不均衡现象，部分数据中心承载的数据量明显高于其他数据中心，从而增加了跨域Shuffle的数据传输开销以及本阶段的计算负载，最终影响任务总体运行时间。相比之下，SDR算法在前一任务阶段对倾斜分区进行了重分区处理，并对大分区进行标记以指导后续任务调度，使得各任务阶段结束后各数据中心的数据量始终保持相对均衡，从而有效降低了后续阶段的通信和计算开销。在数据集1和数据集3中，SDR算法相较于原始分区方案的任务总运行时间分别降低了7.3%和10.2%。综合五个数据集的实验结果，SDR算法相较于原始分区方案平均降低总运行时间 7.8%，明显优于LAHP算法的3.3%和SCID算法的2.3%，体现了其在多阶段任务调度与跨域计算场景下的整体优势。

PageRank任务包含一个循环求解的




(a) 任务阶段执行时间

(b) 任务阶段总运行时间

图7 PageRank第一个Reduce任务阶段时间

操作，会通过循环不断更新贡献度。从第三个Reduce任务阶段开始，后续每一个任务阶段都是重复操作，所以本节实验的PageRank只进行一次更新操作，减少重复的任务阶段。

图12展示了PageRank作业最后一个任务阶段的执行情况，该阶段主要对各页面的贡献度进行汇总，数据格式为 页面编号, 贡献度 。在该阶段中，键值为页面编号，且页面编号在全局范围内呈均匀分布，因此在全局层面上对应的数据分区是均衡的。由于不存在明显的数据倾斜，

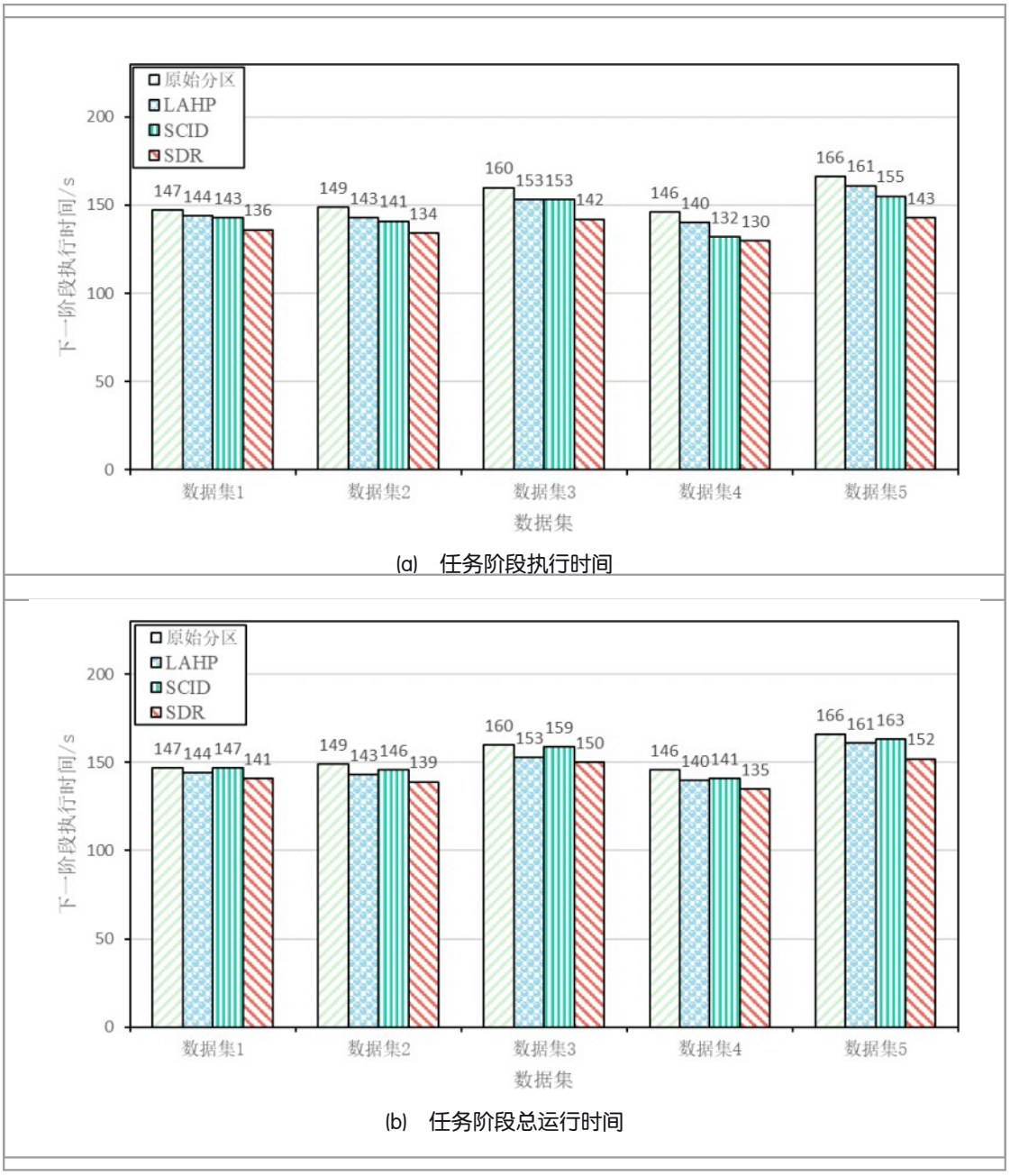



图8 PageRank第二个Reduce任务

该阶段无需执行重分区操作，因此图12中仅给出了任务阶段的总运行时间。

尽管该阶段在全局层面上呈现出均匀分区特性，实验结果仍表明，SDR算法在各个数据集上的任务阶段总运行时间最低。这是因为在前几个任务阶段中，LAHP和SCID算法主要针对单个数据中心内部的数据倾斜进行处理，没有对全局数据分布进行均衡优化，导致部分数据中心在后续任务阶段仍需处理与其计算能力不匹配的数据量，从而增加了跨域数据传输开销。相比之下，SDR算法在每个任务阶段均从全

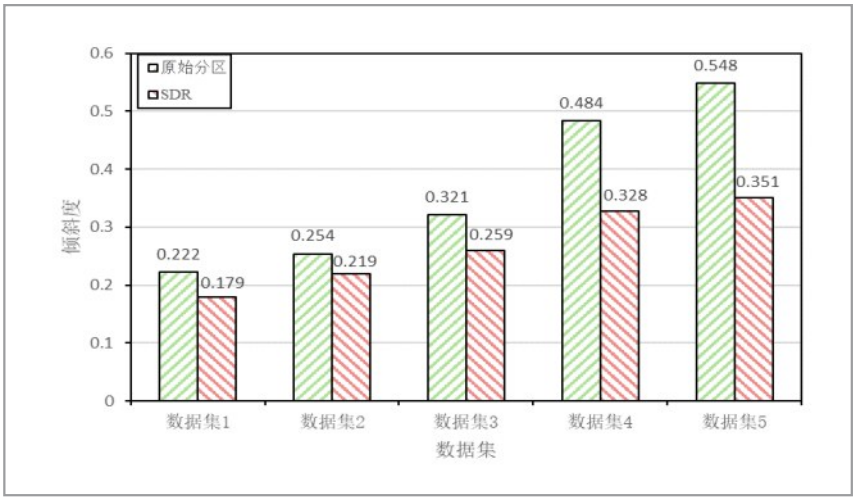


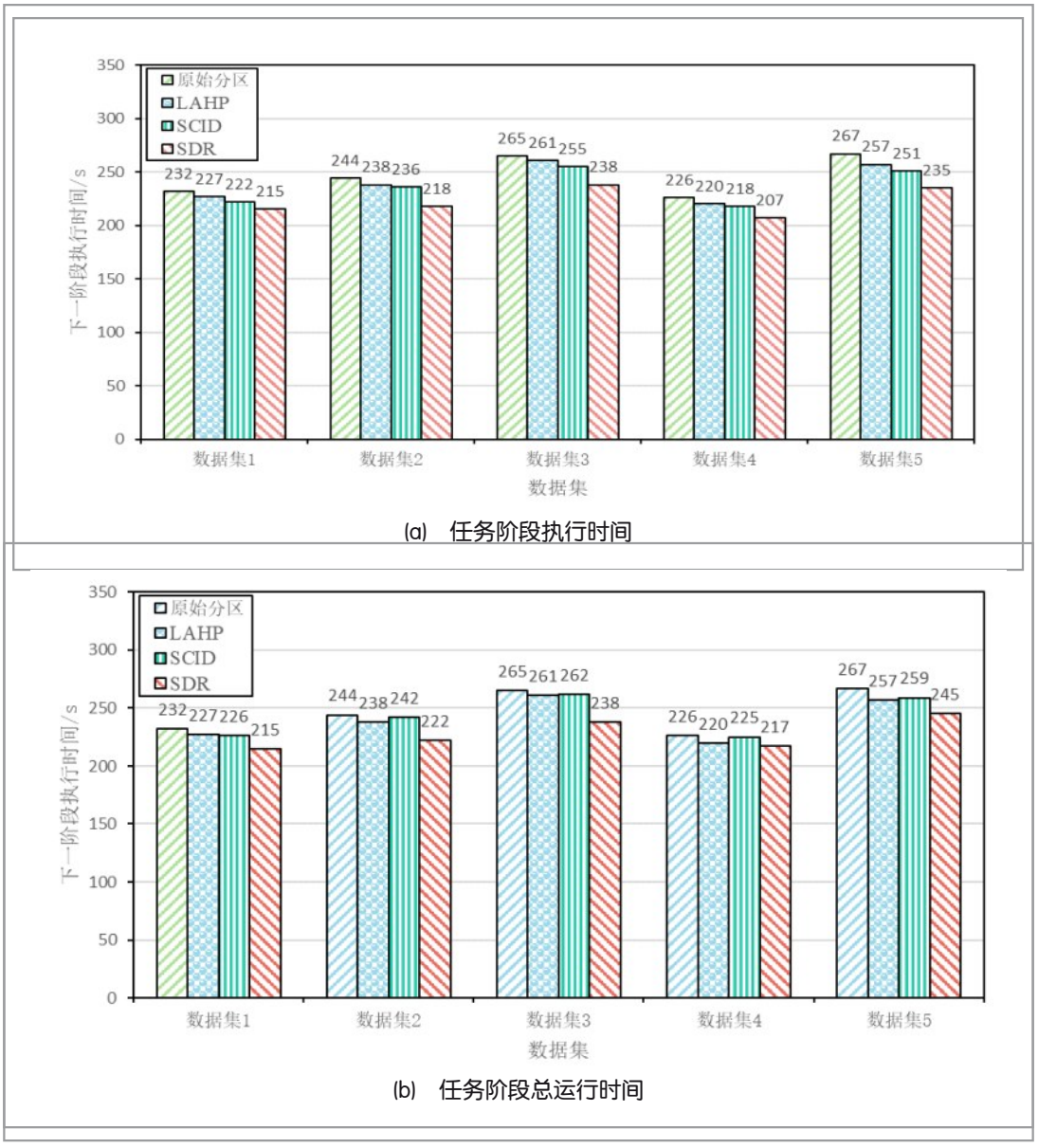

图9 第二个Reduce任务阶段重分区前后倾斜度对比

局角度对各数据中心之间的数据分布进行均衡调整，使各数据中心承担的计算负载与其算力资源更加匹配，从而有效降低了跨域数据传输开销，并减少了任务的总体运行时间。

## 5 结束语

针对跨地域多数据中心大数据计算中普遍存在的数据倾斜问题，本文提出了一种面向跨域场景的数据倾斜检测与均衡重分区方法SDR。首先，针对现有基于均值的倾斜检测方法易受极端分布影响的问题，提出了一种基于分区数据量中位数的倾斜检测策略，并结合key值预聚合机制，在降低跨域通信开销的前提下实现了全局数据倾斜识别。随后，设计了一种保持key一致性的平衡重分区算法，通过贪心策略对倾斜分区进行重划分，使重分区后的分区规模更加均衡。实验结果表明，SDR方法能够在不同倾斜程度的数据集上有效降低分区倾斜度并缩短任务执行时间。

尽管SDR取得了上述效果，仍存在以下局限，需要在未来工作中做进一步研究。在当前算法中，倾斜判定系数$\beta$、$\gamma$以及目标分区容量$C_{target}$均采用固定经验值，这在数据分布相对稳定的场景下能够取得较好的效果，但在面对不同类型任务及多样化数据分布时，其适应性仍然存在一定局限。未来可考虑引入自适应参数调节机制，根据数据分布特征（如分区数据量的方差、偏度或变异系数）以及历史任务执行性能指标（如任务执行时间、资源利用率等），动态调整倾斜检测阈值与目标分区容量，从而使算法能够在不同负载场景下保持稳定的性能表现，提升整体鲁棒性与泛化能力。




(a) 任务阶段执行时间

(b) 任务阶段总运行时间

图10 PageRank第三个Reduce任务阶段时间

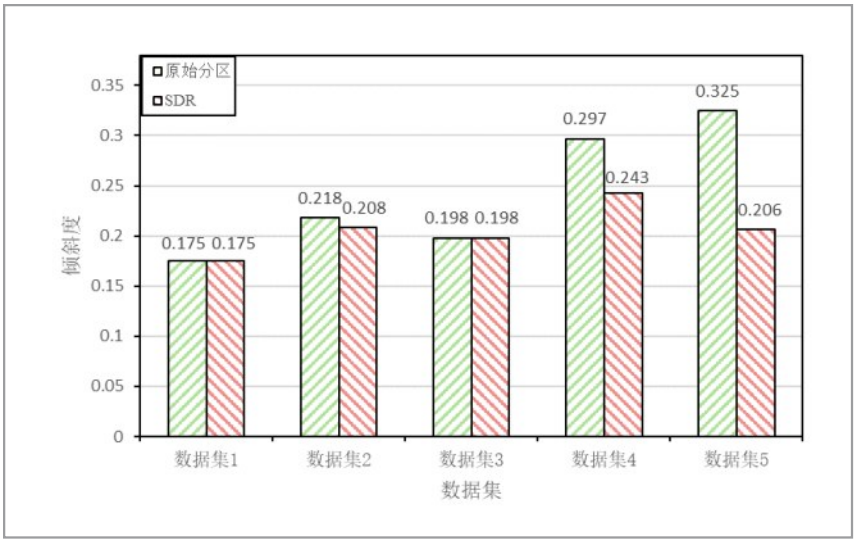



图11 第三个Reduce任务阶段重分区前后倾斜度对比

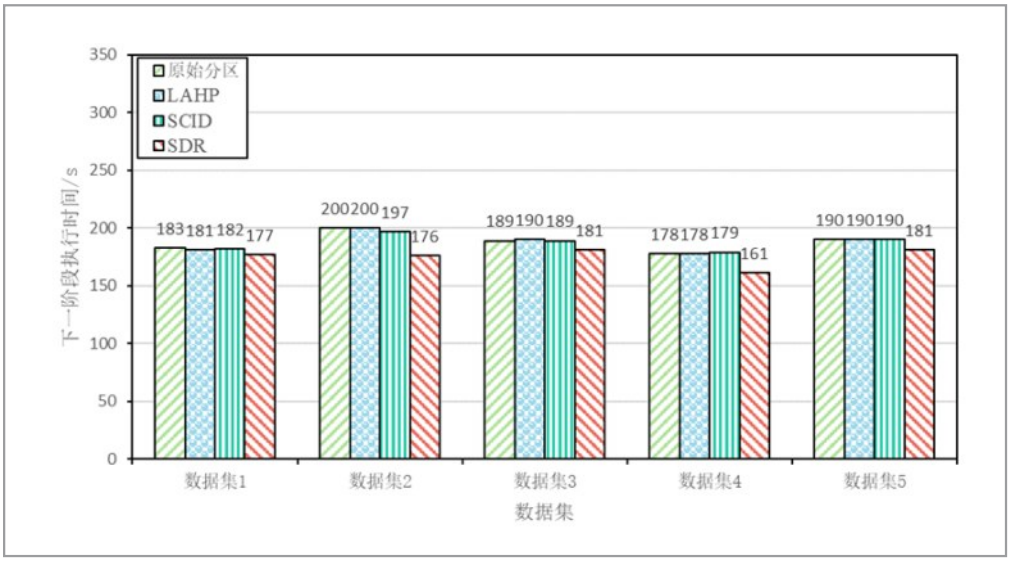
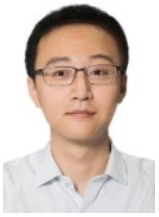



图12 PageRank第四个Reduce任务阶段总运行时间



何玉林(1982- ),男,博士,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,主要研究方向为新型大数据理论模型、大数据高性能/并行计算平台、面向大数据的机器学习算法、复杂型大数据应用系统等。

常家乐(2002- ),女,人工智能与数字经济广东省实验室(深圳)硕士生,主要研究方向为子图匹配、在线学习算法、数据挖掘和机器算法应用等。

徐贺鹏(2000- ),男,深圳大学计算机与软件学院硕士,主要研究方向为大数据分布式计算技术、跨域大数据计算范式、智能大数据处理方法等。

崔来中(1984- ),男,博士,深圳大学大数据技术与应用研究所特聘教授,主要研究方向包括下一代互联网体系结构、软件定义网络、边缘计算、大数据分析、机器学习和智能计算等。

黄哲学(1959- ),男,博士,深圳大学大数据技术与应用研究所特聘教授、所长,主要研究方向为数据挖掘、机器学习、大数据处理与分析、大数据系统计算技术等。

## 参考文献:

[1] 李国杰.大数据与计算模型[J].大数据,2024,10(1):9-16.
LI G J. Big data and computational models[J]. Big Data Research, 2024, 10(1): 9-16.

[2] Phan A C, Phan T C, Cao H P, et al. Comparative analysis of skew-join strategies for large-scale datasets with MapReduce and spark[J]. Applied Sciences, 2022, 12(13): 6554.

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

[4] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster Computing with Working Sets[C]//2nd USENIX Workshop on Hot Topics in Cloud Computing (Hot Cloud). Berkeley: USENIX Association, 2010: 1-7.

[5] HE Z Y, LI Z F, PENG X S, et al. DS2:

Handling Data Skew Using Data Stealings over High-Speed Networks[C]// 2021 IEEE 37th International Conference on Data Engineering (ICDE). Piscataway: IEEE, 2021: 1865-1870.

[6] Kwon Y C, Balazinska M, Howe B, et al. Skewtune: mitigating skew in mapreduce applications[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. 2012: 25-36.

[7] AGARWAL S, DUNAGAN J, JAIN N, et al. Volley: Automated data placement for geo-distributed cloud services[C]// Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2010: 17-32.

[8] Pu Q, Ananthanarayanan G, Bodik P, et al. Low latency geo-distributed data analytics[J]. ACM SIGCOMM Computer Communication Review, 2015, 45(4): 421-434.

[9] Bergui M, Najah S, Nikolov N S. A survey on bandwidth-aware geo-distributed frameworks for big-data analytics[J]. Journal of Big Data, 2021, 8 (1): 40.

[10] 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 generations computer systems, 2024, 152: 138-151.

[11] 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.

[12] 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 (PIC). Piscataway: IEEE, 2015: 102-107.

[13] 何玉林,莫沛恒,Philippe Fournier-Viger等.一种新的以服务质量为导向的Spark作业调度器[J].大数据,2025,11(04):154-177.
HE Y L, MO P H, FOURNIER-VIGER P, et al. A novel quality of service-oriented Spark job scheduler[J]. Big Data Research, 2025, 11(4): 154-177.

[14] Xu Y, Zou P, Qu W, et al. Sampling-based partitioning in MapReduce for skewed data[C]//2012 seventh ChinaGrid annual conference. IEEE, 2012: 1-8.

[15] Tang Z, Zhang X, Li K, et al. An intermediate data placement algorithm for load balancing in spark computing environment[J]. Future Generation Computer Systems, 2018, 78: 287-301.

[16] Fu Z, 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.

[17] Huang C, Tang X. A Data Skew Greedy Optimization Strategy in Spark Heterogeneous Clusters[C]//PARK J S, YANG L T, PAN Y, et al. Advanced Multimedia and Ubiquitous Engineering: AMUE 2024. Lecture Notes in Electrical Engineering, vol. 1475. Singapore: Springer, 2026: 115-122.

[18] Li C, Zhang Y, Luo Y. Intermediate data placement and cache replacement strategy under Spark platform[J]. Journal of Parallel and Distributed Computing, 2022, 163: 114-135.

[19] He Z, Huang Q, Li Z, et al. Handling data skew for aggregation in spark SQL using task stealing[J]. International Journal of Parallel Programming, 2020, 48 (6): 941-956.

[20] 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]. Computer Engineering & Science, 2022, 44(4): 620-630.

[21] Shen Y, Xiong J, Jiang D. SrSpark: Skew-resilient spark based on adaptive parallel processing[C]//2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2020: 466-475.

[22] 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.

[23] Yang L, Xiao X, Zhang X, et al. A real-time partition generation mechanism for data skew mitigation in Spark computing environment[J]. Journal of Grid Computing, 2023, 21(4): 62.

[24] AHN H, KIM H, YOU W. Performance Study of Spark on YARN Cluster Using HiBench[C]//2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia). Piscataway: IEEE, 2018: 206-212.

**录用日期:** XXXX-XX-XX
**通信作者:** 崔来中,cuilz@szu.edu.cn
**基金项目:** 深圳市科技重大专项项目(KJZD20230923114809020);深圳市基础研究面上项目(JCYJ20250604175602004);人工智能与数字经济广东省实验室(深圳)科研任务认领项目(GML-26420011)
**Foundation Items:** Science and Technology Major Project of Shenzhen (KJZD20230923114809020), Basic Research Foundation of Shenzhen (JCYJ20250604175602004), and Research Task Assignment Project from Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ) (No. GML-26420011).