Software Engineering and Applications 软件工程与应用, 2025, 14(2), 217-227
Published Online April 2025 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2025.142020

Hans汉斯

基于MapReduce的数据倾斜优化方法研究

王　涛[1]，王晓玲[1]，王希胤[1,2*]

[1]华北理工大学理学院，河北　唐山
[2]河北省数据科学与应用重点实验室，河北　唐山

收稿日期：2025年2月14日；录用日期：2025年3月27日；发布日期：2025年4月8日

摘　要

本文针对MapReduce框架在处理大规模数据时常见的数据倾斜问题，提出了一种基于抽样的映射分区优化方法。该方法通过水塘抽样算法对数据进行抽样，获取数据分布信息，并结合整体数据分布估计算法和映射分区算法实现数据的均衡分区。实验结果表明，该方法在不同倾斜度下均表现出良好的性能，显著降低了作业执行时间，提高了分区的平衡性，提升了集群资源利用率。

关键词

大数据处理，分布式计算，MapReduce框架，数据倾斜优化

Research on Data Skew Optimization Methods Based on MapReduce

Tao Wang[1], Xiaoling Wang[1], Xiyin Wang[1,2*]

[1]School of Science, North China University of Science and Technology, Tangshan Hebei
[2]Hebei Provincial Key Laboratory of Data Science and Applications, Tangshan Hebei

Received: Feb. 14th, 2025; accepted: Mar. 27th, 2025; published: Apr. 8th, 2025

Abstract

This paper proposes a sampling-based mapping partitioning optimization method to address the common data skew problem in the MapReduce framework when processing large-scale data. The method uses reservoir sampling to sample the data, obtain information on data distribution, and then combines the overall data distribution estimation algorithm and the mapping partitioning algorithm to achieve balanced data partitioning. Experimental results show that the proposed method performs well under different degrees of skewness, significantly reducing job execution

*通讯作者。

文章引用: 王涛, 王晓玲, 王希胤. 基于 MapReduce 的数据倾斜优化方法研究[J]. 软件工程与应用, 2025, 14(2): 217-227.
DOI: 10.12677/sea.2025.142020

time, improving partition balance, and enhancing cluster resource utilization.

Keywords

Big Data Processing, Distributed Computing, MapReduce Framework, Data Skew Optimization

1. 引言

谷歌在大数据领域发挥了引领作用，其发表的三篇标志性论文为大数据技术的发展奠定了基础，也标志着大数据时代的到来。为后续的分布式计算框架提供了理论基础，也为其他行业和研究领域的发展带来了新的力量。其中关于 MapReduce 论文提出了分布式计算的编程模型[1]，极大地简化了大规模数据处理的复杂性，提高了集群对大规模数据的处理能力。MapReduce 作为一种经典的分布式计算框架，主要应用于大规模数据处理任务中。并且，广泛应用于多个领域，其中包括文本处理、网络分析，数据挖掘，生物信息和机器学习等。然而，有的时候 MapReduce 框架会面临数据倾斜问题，这不仅降低了计算效率，还可能导致资源浪费和集群性能下降。

MapReduce 作为一种高效的分布式计算框架，能够有效处理大规模数据集。通过优化 MapReduce 框架中的分区方法，可以有效减少倾斜数据的产生，提高集群的计算性能。优化数据倾斜可以避免部分节点负载过重而其他节点相对空闲的情况，使集群资源得到更合理的分配和利用，充分发挥集群的计算能力，提高整体的作业处理效率。数据倾斜会导致某些节点的负载过高，从而增加系统的压力，可能导致系统崩溃或性能下降。优化数据倾斜可以有效缓解这种压力，使系统运行更加稳定。

2. 研究现状

MapReduce 的默认分区器 HashPartitioner，通过对键进行哈希来确定记录所属的分区以及因此所属的 Reduce 任务，分区数量等于作业的 Reduce 任务数量[2]。然而这种方法存在一定的局限性，它在面对大量有哈希冲突的键时，无法均匀地将键分到分区中。基于此，很多文献提出了各自的应对方法。Tang 等人从数据局部性和通信成本的角度出发，提出了一种基于采样的 Reduce 任务放置算法(CORP)，通过分析中间结果的分布矩阵，优化 Map 和 Reduce 任务的物理节点分配，减少了跨节点通信，提高了系统性能[3]。Rivault 等人针对轨迹数据的相似性连接问题，提出了一种基于 MapReduce 和局部敏感哈希的可扩展相似性连接算法[4]。Suguna 等人则在云计算环境中提出了一种动态云分区和负载平衡的策略[5]。Elaheh Gavagsaz 等人提出了一种新的负载均衡方法，用于处理 MapReduce 系统中因数据倾斜导致的连接算法性能下降问题。该方法称为细粒度分区方法(FGSD)，通过流采样算法同时考虑输入和输出数据的属性，优化了数据在 reduce 任务中的分布[6]。Elaheh Gavagsaz 等人提出了一种基于可扩展简单随机抽样的负载均衡方法。该方法通过 ScaSRS 算法对中间数据进行采样，估计键的频率分布，并据此制定数据放置策略。通过排序平衡算法 SBaSC，该方法能够显著提高 reduce 任务的负载均衡水平，并减少执行时间[7]。Tao Gao 等人提出了一种针对超级计算系统的内存高效且倾斜容忍的 MapReduce 框架 Mimir。该框架通过引入组合器优化、动态重分区方法和超级键分割策略，有效平衡了内存使用，并显著提高了处理倾斜数据集的性能[8]。Lei Chen 等人提出了一种基于朴素贝叶斯分类器的分区器 BAPM，用于优化 MapReduce

框架中的数据局部性和数据倾斜问题。BAPM 通过自动选择合适的分区算法，根据作业类型和带宽条件，动态调整数据分配策略，从而提高 MapReduce 作业的执行效率[9]。Zhuo Wang 等人提出了一种增量分配方法，用于减少 MapReduce 中的分区倾斜问题。该方法通过将映射数据划分为多个微分区，并在映射过程中逐步收集这些微分区的大小统计信息，然后在多个轮次中将微分区增量分配给归约器[10]。Zhihong Liu 等人提出了一种运行时动态资源调整技术 Dynamic Adjust，用于缓解 MapReduce 中的数据倾斜问题。Dynamic Adjust 通过在运行时监控任务的执行情况，并动态增加计算资源来缓解倾斜问题，从而消除了负载预测和重新平衡的开销[11]。Daikoku 等人提出了一种针对 MapReduce 洗牌阶段的内存中混洗方法。他们设计并评估了三种洗牌架构：耦合洗牌架构(CSA)、解耦洗牌架构(DSA)以及带有倾斜感知元洗牌的解耦混洗架构[12]。张元鸣、蒋建波、陆佳炜、徐俊和肖刚等人提出了一种面向 MapReduce 的迭代式数据均衡分区策略，该策略通过将每个 MapReduce 节点处理的数据块细分为微分区，并以迭代方式循环处理，根据已迭代轮次的微分区分配结果动态调整当前轮次的分配方案，逐步实现数据均衡分区[13]。

3. 数据倾斜

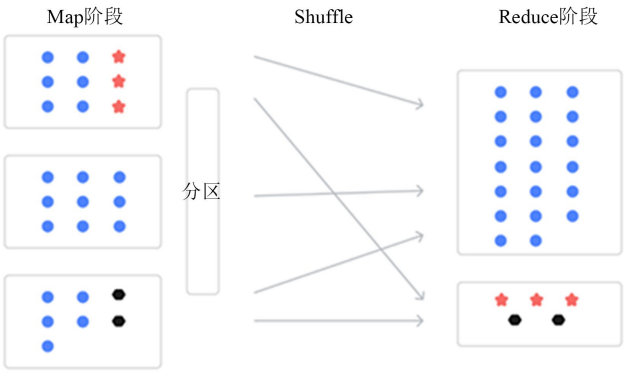

Figure 1. Schematic diagram of data skew
图 1. 数据倾斜示意图

如图 1 所示，是一种数据倾斜情形，图中不同形状代表不同的数据。前面在介绍理论知识部分提到过 MapReduce 计算框架主要分为两部分 Map 和 Reduce。数据存储时 HDFS 会将数据分块存储，而在计算过程中会将数据划分成片，Map 阶段会按照“分片”为最小计算单元进行计算如图 2 所示，每个分片可以理解为一个线程所处理的数据。

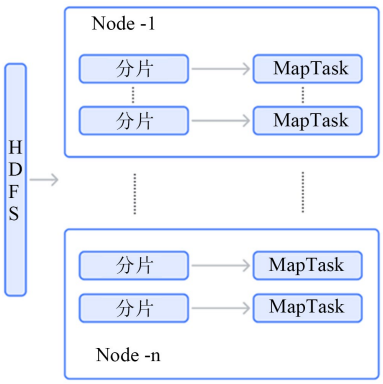

Figure 2. Schematic diagram of “sharding”
图 2. “分片”示意图

内存中处理完的数据会先写入一个逻辑存储空间—环形缓存区中，环形缓冲区中的数据如果达到一个预设的阈值会将多个 Map 的结果进行一个初始聚合操作。最终将多个合并后的文件再进行最终的合并，合并后的结果会通过哈希函数映射到预定的分区中，由于哈希冲突的原因，多个相同哈希结果的会被映射到同一个分区，随着映射到某一个分区的数据越来越多，就形成了数据倾斜。因为往往 Reduce 的个数和分区个数默认是相同的，而 Reduce 拉取对应分区的数据进行计算。通常一个节点只有一个 Reduce 任务，所以数据倾斜会导致单个节点的负载过重，影响总体任务处理进度，从而降低集群的性能。数据倾斜可能和数据集中存在热点键，以及数据处理过程中的逻辑有关。

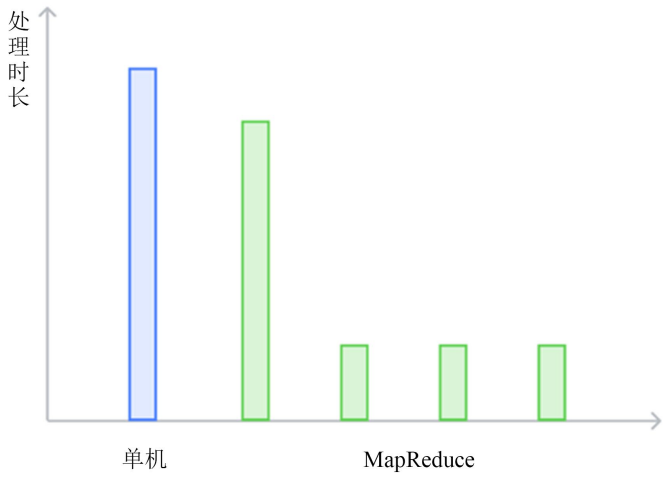

Figure 3. The performance impact of data skew
图 3. 数据倾斜的性能影响

数据倾斜的影响主要表现在两方面，一是性能下降，二是资源利用率低，如图 3 所示。由于部分 Reduce 任务需要处理大量的数据，而其他任务可能很快完成，整个 MapReduce 作业的执行时间会被拉长，因此整个数据的处理时间和单机差不多，甚至比单机处理的还要久。因为在 MapReduce 中，作业的完成时间通常取决于最慢的 Reduce 任务，而一个 MapReduce 任务没有结束会影响下一个任务的开始。例如，在一个数据仓库的数据加载作业中，如果存在数据倾斜，可能会使原本几分钟就能完成的任务延长到几十分钟甚至几个小时，严重影响数据处理的时效性。不仅会影响性能，还可能产生资源闲置，造成资源利用率低。处理大量数据的 Reduce 任务会占用过多的资源，如 CPU、内存和网络带宽等。而其他任务所在的节点资源可能处于闲置状态，导致集群资源的利用率低下。例如，在一个云计算环境下的 MapReduce 集群中，因为数据倾斜，部分服务器的 CPU 利用率可能达到 100%，而其他服务器 CPU 利用率很低，随着倾斜度越高资源的利用率越低。所以解决数据倾斜问题显得尤为重要，优化解决数据倾斜的策略需要不断深入的研究。

4. 映射分区

4.1. 方法概述

前面介绍了数据倾斜主要是 MapReduce 框架在分区的时候采用哈希算法来将大量的中间数据映射到同一个分区所导致的，在应对的时候有以下方法：

1) 在数据处理前给每个数据加上一个随机的标签也称作加盐，这样计算出的哈希值也较为随机，可以避免大量的哈希冲突。

2) 尽量使 ReduceTask 在产生数据量较大的节点上运行，避免大量的网络开销。

3) 范围分区法。对于有明显特征的中间键 key，例如：年龄，时间等，采用范围分区。比如：10~20 岁在一个分区，20~40 岁在一个分区等。

这些方法虽然可行，但都有他们的局限性。比如，加入随机数的操作需要遍历所有数据，当数据量过大时这样做会浪费计算资源。产生中间数据量较大的节点运行 Reduce Task，虽然避免了网络的开销减少了运行时间，但在面对数据倾斜时，并没有平衡分区。单纯的范围分区无法感知数据分布情况，某个范围的数据可能很大。

本文总结了这些方法的缺点，并提出了一种提出了应对数据倾斜的方法。该方法主要分为两部分：首先对数据集进行抽样，将抽象出来的数据组成新的数据集，然后将新的数据集进行同样的作用任务进行计算，得出 Map 后的中间数据，从而预先感知原有整体数据的 key 的分布情况。最后，根据分布情况制定预分区策略，采用映射分区算法将数据均匀地映射到各分区中。

4.2. 抽样方法

抽样是从目标总体中选择一个代表性样本并从该样本收集数据，以便对总体有一个整体的了解的过程。可以通过抽样出来的小样本来估计目标整体样本的数据分布情况。为了有效解决数据倾斜问题，选择合适的抽样方法是必不可少的。大数据集的数据量庞大，且可以包含多种类型数据，数据来源也可能很多。而水塘抽样是用于从大规模数据流中随机抽取固定数量样本的算法，特别适用于数据量非常大的场景。水塘抽样的步骤如下：

1. 初始化水塘：将数据流中的前 k 个元素直接放入水塘中，其中 k 是要抽取的样本数量。

2. 遍历剩余元素：对于数据流中的第 i 个元素($i > k$)，执行以下操作：

(1) 生成一个随机数 j，范围是 1 到 i (包括 i)。

(2) 如果 $j \leq k$，则用第 i 个元素替换蓄水池中的第 j 个元素。

(3) 如果 $j > k$，则不替换，继续处理下一个元素。

3.返回结果：遍历完数据流后，水塘中的 k 个元素即为最终的样本。

水塘抽样算法伪代码如下表 1 所示。

Table 1. Reservoir sampling algorithm
表 1. 水塘抽样算法

算法 1 水塘抽样算法

```
输入：总体数据 M，样本大小 n
输出：大小为 n 的样本 S
1:   S ← initialized array // size of n
2:   i ← 0
3:   for each s in M do
4:       if i ≤ n then
5:           S[i] ← s
5:       else
6            j ← random(1,i)
7:           if j ≤ n then
8:               S[j] ← s
9:           end if
10:      end if
11:      i ← i+1
12: return S
```

水塘抽样算法能保证每个元素被选中的概率相等，即：$\frac{k}{n}$，其中 n 是目标整体。

在 MapReduce 中实际使用水塘抽样的时候，创建一个自定义的采样器类，可以通过自定义 InputFormat 或 RecordReader 来实现采样器。在该类中，实现采样逻辑，包括初始化采样器、读取数据并进行抽样。定义完采样器后需要配置 MapReduce 作业，通过 Job 类的 setInputFormatClass 方法，将自定义的采样器类设置为输入格式类。这样，在作业运行时，采样器会先对输入数据进行抽样。选择需要抽样的样本大小，使用水塘抽样算法进行抽样。然后对样本数据进行 Map 和 Reduce 运算，得到样本的中间数据 $S=\{(k1,v1),(k2,v2),\cdots,(kn,vn)\}$，$n\in(1,2,3,\cdots)$。其中 key($k$)和 value($v$)分别为键和值，是 Map 阶段的输出，也是 Reduce 阶段的输入，通过 key 可以了解数据的分布情况。

4.3. 分区方法

前面小结我们通过抽样方法抽样出部分中间数据，并得出中间数据的二维矩阵。有了中间数据的二维矩阵后可以估计整体的数据分布情况。本文提出了整体数据分布情况估计算法 GDDEA。该算法通过中间数据样本矩阵 Sample matrix 和给定的抽样率 R 推断整体数据分布。首先初始化一个集合 S，用于存储唯一元素及其计数。随后，遍历样本矩阵中的每个键值，若键已存在于集合 S 中，则将其计数加一；若不存在，则将该键及其计数 1 添加到集合中。完成计数后，再次遍历集合 S，将每个元素的计数除以抽样率 R，以估计其在整体数据中的分布频率。最终返回集合 S，其中包含整体数据分布的估计值。表 2 是该算法的伪代码。

Table 2. Overall data distribution estimation algorithm
表 2. 整体数据分布情况估计算法

算法 2 整体数据分布情况估计算法 GDDEA

输入：中间数据 Sample matrix，抽样率 R
输出：整体数据分布集合
1: S ← ∅ // elements are unique
2: for each key in Sample matrix do
3: if key in S then
4: S_{key} ← <key:S.key.num+1>
5: else
6: S_{key} ← <key:1>
7: end if
8: end for
9: for each S_i in S do
10: S_i ← <key_i:num/R>
11: end for
12: return S

有了整体中间数据的分布集合 S 后，根据分布集合 S 合理分配数据到不同的分区。本文提出了一种映射分区算法 ATMP，以优化数据处理效率并缓解数据倾斜问题。ATMP 算法的输入为分布集合 S、分区个数 n 和倾斜分区的占比阈值 T。该算法首先计算整体数据的总数 total，随后遍历集合 S 中的每个元素 Si。对于每个元素 key 的值，若其占比 $\frac{S_i\cdot num}{total}$ 超过阈值 T，则将该元素进一步拆分为 n 个子元素，并为每个子元素分配一个独立的分区标识符 PartitionID，存储于集合 P 中。否则，直接根据哈希函数计算 key 的哈希值，然后将其对分区总数 n 取模后得到分区值。最后，ATMP 算法返回包含所有映射关系的集合 P，为后续均衡分区提供预分区映射关系。表 3 是映射分区算法 ATMP 的伪代码。

Table 3. Mapping partitioning algorithm
表 3. 映射分区算法

算法 3 映射分区算法 ATMP
输入：整体数据分布集合 S，分区个数 n，倾斜分区的占比阈值 T 输出：映射分区集合 1: total ← 0// initialize to zero 2: P ← ∅// elements are unique 3: for each S_i in S do 4: total ← S_i.num+total 5: for each S_i in S do 6: if S_i.sum/total ≥ T then 7: $S_i \rightarrow \{S_{i1},S_{i2},..,S_{in}\}$ 8: for each S_j in S_i do 9: P_j ← <key_j:PartitionID> 10: end for 11: else 12: P_i ← <key: hash(Si.key)%n> 13: end if 14: end for 15: return P

在对数据处理的时候，可以根据映射分区集合，将数据映射到对应的分区中。首先在自定义分区器的时候需要继承 Partitioner 类，以映射分区元组 P 作为输入，输出为对应的分区 ID。其核心逻辑基于对映射分区元组的逐项遍历与匹配操作。具体而言，对于输入键 k，依次检查其是否与元组 P 中的每个分区元组 Pi 的键匹配。若匹配成功，算法进一步判断 Pi 是否将元素拆分到多个分区中，即检查 Pi 的长度是否大于 1。若 P_i 包含多个分区，则算法继续遍历其子分区 P_j，并返回匹配分区的 PartitionID。反之，若 Pi 未进行拆分，则直接返回 Pi 的 PartitionID。若输入键 k 未匹配到任何分区元组，则算法采用哈希函数计算 key 的哈希值，然后对分区总数取模，生成分区 ID，以确保未匹配键能够被合理分配到分区中。表 4 是数据映射算法的伪代码。

Table 4. Data mapping
表 4. 数据映射

算法 4 数据映射
输入：映射分区元组 P 输出：分区 ID 1: extends Partitioner<k,v> 2: for each P_i in P do 3: if <k match P_i.key> is true then 4: if P_i.length > 1 then 5: for each P_j in P_i do 6: return P_j.key.PartionID 7: else 8: return P_i.key.PartionID 9: end if 10: else 11: return k.HashCode%PartionNum 12: end if 13: end for

5. 倾斜模型

为了量化使用分区方法后，不同 Reduce 任务的倾斜程度，通过标准差的方式来量化整体的倾斜度。

后续实验也可以通过此方法来衡量分区方法的平衡分区的能力。量化方式如下：

设 T_i 为第 i 个 Reduce 任务的处理数据量，T_{total} 为所有 Reduce 任务的总处理数据量。则单个 Reduce 任务的倾斜度 S_i 可以表示为：

$$S_i = \frac{T_i}{T_{total}} \tag{1}$$

设 T_{avg} 为所有 Reduce 任务的平均处理数据量，即：

$$T_{avg} = \frac{T_{total}}{n} \tag{2}$$

其中 n 为 Reduce 任务的总数。

整体倾斜度 S_d 可以用方差来表示，即每个 Reduce 任务的处理数据量与平均处理数据量之差的平方和除以任务总数 n：

$$S_d = \frac{1}{n}\sum_{i=1}^{n}\left(T_i - T_{avg}\right) \tag{3}$$

S_d 的值越小表示倾斜程度越低，分区方法的性能越好。

6. 实验

6.1. 实验方案

1) 实验目的

本文旨在通过实验，验证所提出的映射分区算法 ATMP 在不同抽样率、倾斜程度、阈值以及真实热点键数据情况下的性能表现，并与其他常见分区方法进行对比分析，以评估该算法在数据分区效率及平衡性方面的优势与适用性。

2) 实验环境

本实验包含一个主节点和六个从节点，集群服务器采用 Hadoop3 版本。通过免密登录协议，实现了服务器之间的免密通信，从而高效便捷地支持数据传输。

3) 数据集

数据是合成数据集，该数据集生成了大量遵循 Zipf 分布的文本数据。Zipf 是一个描述自然语言中单词频率分布的数学模型。它表明在任何足够大的自然语言文本中，一个单词的频率与它在频率表中的排名成反比。Zipf 公式可以表示为：

$$f_{(k,N,s)} = \frac{1/k^{\alpha}}{\sum_{n=1}^{N}\left(\frac{1}{n^{\alpha}}\right)} \tag{4}$$

6.2. 采样实验

由于本文提出的映射分区算法涉及到抽样，而抽样需要确定抽样率，也就是样本量占总体的比重。这里选用了 5%，10%，15%，20%和 25%的抽样率进行对比实验。对每个倾斜度的数据集，分别采用上述五种抽样率进行抽样。在每种抽样率下，运行映射分区算法，记录每次运行的执行时间。通过对比不同抽样率下各倾斜度数据集的执行时间，分析抽样率对算法性能的影响。预期随着倾斜度的增加，不同抽样率下的执行时间均会增加，但某些抽样率下执行时间的增长趋势可能较为平缓，从而确定在不同倾斜程度下较为合适的抽样率范围。实验结果如下。

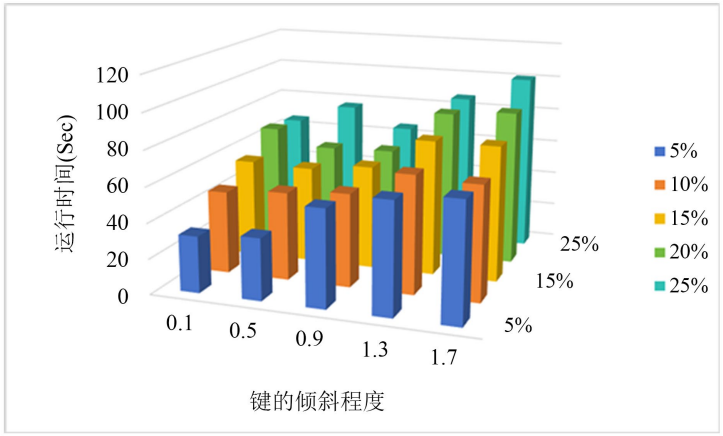

Figure 4. Execution time under different sampling rates and different degrees of skewness
图 4. 不同抽样率和不同倾斜度的执行时间

图 4 是抽样率分别为 5%，10%，15%，20%和 25%的情况下使用算法进行分区对执行时间的影响。从图中可以看出随着倾斜度的增加，不同抽样率下，执行时间都在增加。10%和 15%抽样率下，随着倾斜度的增加执行时间的增长较为平缓。在倾斜度为 0.1 和 0.5 的时候，抽样率为 5%的效果最好，作业执行时间最少。但是随着倾斜程度的越来越高，5%下的执行时间越来越多，倾斜度为 0.9 的时候，几种抽样率下执行时间相差不大。而倾斜度在 1.3 和 1.7 的时候，几种抽样率的效果都比较差。抽样率在 20%和 25%的时候，执行时间的波动性比较大。综合来看 10%和 15%的抽样率下，随着倾斜度的增加执行时间增长较为平缓。而 10%的抽样下，总体执行时间较短，所有后续实验中采样 10%的抽样较为合适。

6.3. WordCount 实验

这里使用常见的词频统计来测试本文提出的分区方法的性能，并与其他几种分区方法进行比较。对每个倾斜程度的数据集，分别采用映射分区算法 ATMP、默认的哈希分区算法 HP、Range 分区方法和随机加前缀分区方法 RP 进行分区操作。对比不同分区方法在不同倾斜程度下的执行时间，分析各方法在应对数据倾斜时的性能表现，以及平衡分区能力的差异，从而判断映射分区算法在实际操作时的性能。实验结果为倾斜度从 0.1 等比例增到 1.7 时不同分区方法的执行时间的长短和平衡分区的能力。

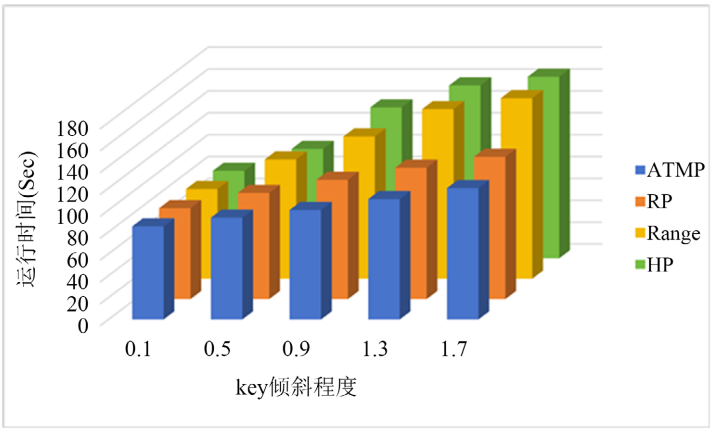

Figure 5. Execution time of four algorithms under different skew levels
图 5. 不同倾斜程度下四种算法的执行时间

图 5 展示了 4 种算法的运行时间结果对比图。可以发现，在倾斜程度为 0.1 的时候，四种方法的运行时间几乎差不多，其中默认的哈希分区算法的运行时间最快，这是因为哈希分区算法 HP 实现起来相对较为简单，并不涉及到抽样等其他相关操作。在倾斜程度较低时选择默认的哈希分区算法进行分区的运行时间较优。然而，随着倾斜程度的增加，HP 未考虑到数据的分布情况，所以运行时间比 ATMP 和 RP 的要多，并且运行时间的增长也较快。Range 分区方法的运行时间也越来越快且增长速度较快，这是因为 Range 分区方法需要明确了解哪些范围的数据较多，如果没有明确范围，使用 Range 分区的运行时间反而更多。而 ATMP 和 RP 方法的运行时间增长较为缓慢，ATMP 方法的运行时间甚至比 RP 方法的更短。

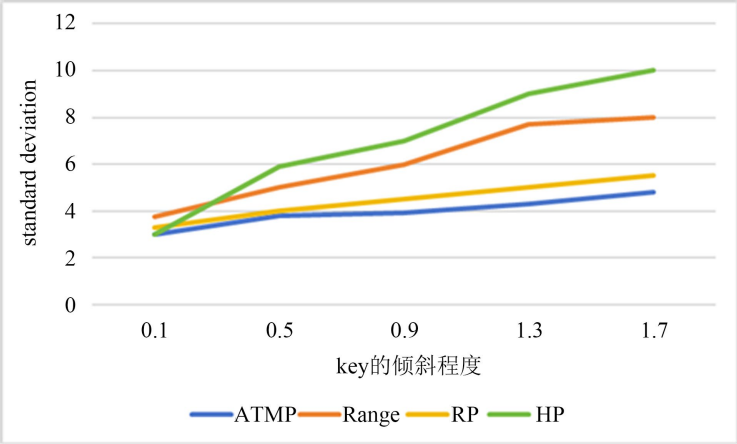

Figure 6. Standard deviations of the four algorithms under different skew levels
图 6. 不同倾斜程度下四种算法的标准差

图 6 展示了四种方法的平衡分区的能力。可以看出这四种方法随着 key 的倾斜程度的增加，标准差值都有着不同程度的增加，说明各个分区中数据量差异程度增大。但其中默认的哈希分区方法 HP 和 Range 分区方法的标准差值增长程度较大，不同分区中数据量差异较大，平衡分区能力不理想。而 ATMP 和 RP 分区方法的标准差虽然也在增加，但是增幅较小，甚至 ATMP 的平衡分区的性能更优。

7. 结论

本章围绕数据倾斜问题展开深入研究，系统地探讨了数据倾斜的成因、影响及其优化方法。首先，通过分析 MapReduce 框架的运行机制，揭示了数据倾斜的根源在于哈希冲突导致的分区不均衡，进而阐述了数据倾斜对系统性能的负面影响，包括任务处理时间延长、资源利用率低下以及热点键问题。为解决这一问题，本文提出了一种基于抽样的映射分区算法。通过水塘抽样算法对数据进行抽样，以获取数据分布的先验信息，进而利用整体数据分布估计算法 GDDEA 和映射分区算法 ATMP 实现数据的均衡分区。实验部分从多个维度对所提方法进行了验证，包括不同抽样率、倾斜程度等。结果表明，所提出的映射分区算法在执行时间和平衡分区能力方面均优于传统方法，特别是在高倾斜度数据集上展现出显著的性能优势。

参考文献

[1] Dean, J. and Ghemawat, S. (2008) MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, **51**, 107-113. https://doi.org/10.1145/1327452.1327492

[2] Fan, Y., Wu, W., Xu, Y. and Chen, H. (2014) Improving Mapreduce Performance by Balancing Skewed Loads. *China*

Communications, **11**, 85-108. https://doi.org/10.1109/cc.2014.6911091

[3] Zhuo, T., Wen, M., Kenli, L., *et al.* (2016) A Data Skew Oriented Reduce Placement Algorithm Based on Sampling. *IEEE Transactions on Cloud Computing*, **8**, 1149-1161.

[4] Rivault, S., Bamha, M., Limet, S. and Robert, S. (2022) A Scalable Similarity Join Algorithm Based on MapReduce and LSH. *International Journal of Parallel Programming*, **50**, 360-380. https://doi.org/10.1007/s10766-022-00733-6

[5] Suguna, R., Divya, M. and Ranjani, R. (2014) A Novel Approach for Dynamic Cloud Partitioning and Load Balancing in Cloud Computing Environment. *Journal of Theoretical and Applied Information Technology*, **62**, 662-667.

[6] Gavagsaz, E., Rezaee, A. and Haj Seyyed Javadi, H. (2018) Load Balancing in Join Algorithms for Skewed Data in Mapreduce Systems. *The Journal of Supercomputing*, **75**, 228-254. https://doi.org/10.1007/s11227-018-2578-0

[7] Gavagsaz, E., Rezaee, A. and Haj Seyyed Javadi, H. (2018) Load Balancing in Reducers for Skewed Data in MapReduce Systems by Using Scalable Simple Random Sampling. *The Journal of Supercomputing*, **74**, 3415-3440. https://doi.org/10.1007/s11227-018-2391-9

[8] Gao, T., Guo, Y., Zhang, B., Cicotti, P., Lu, Y., Balaji, P., et al. (2020) Memory-efficient and Skew-Tolerant MapReduce over MPI for Supercomputing Systems. *IEEE Transactions on Parallel and Distributed Systems*, **31**, 2734-2748. https://doi.org/10.1109/tpds.2019.2932066

[9] Chen, L., Lu, W., Bao, E., Wang, L., Xing, W. and Cai, Y. (2018) Naive Bayes Classifier Based Partitioner for MapReduce. *IEICE Transactions on Fundamentals of Electronics*, *Communications and Computer Sciences*, **101**, 778-786. https://doi.org/10.1587/transfun.e101.a.778

[10] Wang, Z., Chen, Q., Suo, B., Pan, W. and Li, Z. (2019) Reducing Partition Skew on MapReduce: An Incremental Allocation Approach. *Frontiers of Computer Science*, **13**, 960-975. https://doi.org/10.1007/s11704-018-6586-2

[11] Liu, Z., Zhang, S., Liu, Y., Wang, X. and Yin, D. (2021) Run-Time Dynamic Resource Adjustment for Mitigating Skew in MapReduce. *Computer Modeling in Engineering & Sciences*, **126**, 771-790. https://doi.org/10.32604/cmes.2021.013244

[12] Daikoku, H., Kawashima, H. and Tatebe, O. (2019) Skew-aware Collective Communication for MapReduce Shuffling. *IEICE Transactions on Information and Systems*, **102**, 2389-2399. https://doi.org/10.1587/transinf.2019pap0019

[13] 张元鸣, 蒋建波, 陆佳炜, 等. 面向 MapReduce 的迭代式数据均衡分区策略[J]. 计算机学报, 2019, 42(8): 1873-1885.