西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

《大数据技术原理与应用》

理论报告

2023-2024 学年第 2 学期

姓　　名	：	张苏宇
学　　号	：	**2021302853**
班　　级	：	**14012106**
学　　院	：	软件学院
任课教师	：	田春伟

目 录

基于改进 SVD++算法和 K-means++算法的小文件合并方案

张苏宇
西北工业大学软件学院

摘　要：文章提出了一种基于改进 SVD++算法和 K-means++算法的小文件合并方案。通过引入自适应学习率函数和基于并行分组的 SVD++算法，优化了小文件的合并过程，以提高 Hadoop 存储小文件的效率。同时，利用 K-means++算法对合并后的文件进行聚类，优化了数据存储方式，降低了存储空间的浪费。在 Hadoop 平台上进行的实验表明，该方案在保持数据处理准确性和稳定性的同时，显著提升了 Hadoop 存储与处理小文件的性能。

关键词：Hadoop;小文件合并;SVD++算法;K-means++算法。

0 引言

随着大数据应用的不断拓展和普及，Hadoop 作为一种分布式计算和存储框架，在处理海量数据方面发挥着重要作用。然而，Hadoop 在处理大量小文件时面临着存储效率低下和计算性能下降的问题。小文件数量庞大且分散存储，导致了存储空间的浪费和数据处理的低效。为了应对这一挑战，许多研究者对小文件合并技术进行了深入探讨，旨在优化 Hadoop 存储小文件的效率[1]。

1 相关技术研究

小文件合并是解决 Hadoop 处理海量小文件效率低下的重要问题。在现有研究中，Hadoop Archive (HAR)、SequenceFile 和 Mapfile 等[2]是常见的基础方法用于小文件合并。

HAR 方案：HAR 方案是 Hadoop 官方提供的一种文件合并方案。该方案通过将小文件打包成 HAR 文件，从而减少了 NameNode 的负载，降低了存储空间的浪费。虽然 HAR 方案在一定程度上改善了小文件处理效率，但由于只能在文件级别进行打包，可能导致 HAR 文件仍然较小，限制了其在解决大量小文件问题上的应用。

SequenceFile: SequenceFile 是 Hadoop 提供的二进制键值对文件格式，可以将小文件合并成一个大文件。虽然 SequenceFile 可以较好地解决小文件问题，但其不支持数据块的拆分和索引，导致在大规模数据处理中可能出现数据倾斜等问题，影响了数据的读取性能。

Mapfile: Mapfile 是另一种用于小文件合并的格式，它支持数据块的拆分和索引，可以提高数据读取的效率。然而 Mapfile 对于小文件的合并并不是特别高效，尤其在处理大量小文件时，可能仍然存在性能瓶颈。

除了上述基础方法，还有一些其他的小文件合并方法在学术界得到了探索。例如，根据“文件类型”(文本、文档、二进制等)和“文件大小”(文件所需的存储空间)来过滤文件，同时采用动态合并技术对小文件进行合并[3];利用用户协同过滤和缓存机制的方法[4]可以提高小文件的存取效率，实现更智能化的小文件合并策略。

综上所述，虽然已有多种小文件合并方法，但每种方法都存在一定的局限性。本文提出的基于改进 SVD++算法和 K-means++算法的小文件合并方案旨在综合利用不同方法的优点，通过自适应学习率函数和并行分组等技术，优化合并过程，并借助 K-means++算法进一步优化存储方式，从而提高 Hadoop 存储与处理小文件的效率。

2 SVD++算法及其改进

2.1 SVD++算法

在研究 Hadoop 平台存储小文件的改进方案时，合并小文件是一种良好的解决方案。然而如何确定哪些小文件应该被合并是一个挑战。传统的基于文件大小或时间戳的合并方法可能会导致合并后的文件过大或过小，从而影响数据处理和存储效率。基于矩阵分解的推荐算法是一类通过对用户-物品评分矩阵进行分解，从而得到用户和物品的潜在特征向量，以预测未知评分并进行推荐的算法。SVD++算法就是一种基于矩阵分解的推荐算法，是对传统 SVD 算法的改进。SVD++算法的优势在于不仅可以考虑文件之间的相关性，而且还可以考虑用户的个性化需求。它可以将不同用户之间的访问记录区分开来，并针对每个用户生成相应的小文件合并方案。这样可以更好地满足用户的需求，同时提高数据处理和存储的效率。在本文的研究中，可以将小文件视为物品，将用户的访问记录视为评分。通过将小文件的访问记录转换为向量表示，则可以利用 SVD++算法来找到小文件之间的关联性。SVD++算法具体的预测公式可以表示为：

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j) \tag{1}$$

其中，r 是用户 u 对小文件 i 的预测评分，μ是全局平均评分，bi 和 bu 分别表示用户偏置和小文件的偏置，qi 表示小文件的隐含特征向量，pu 表示用户 u 的隐含特征向量，N(u)表示用户访问过的小文件的集合，yj 表示用户 u 的隐式反馈。

但 SVD++算法仍然存在着一些不足[5]:

(1) SVD++算法的计算复杂度较高，因为它需要对整个数据集进行分解和求解。这在大规模数据集中会导致计算时间和资源的浪费。

(2) SVD++算法在处理稀疏数据时也存在问题，因为它只能对已有的评分数据进行分解，而对于缺失数据的处理比较困难。因此，需要对 SVD++算法进行一定的改进以应对大数据背景下对海量小文件的处理。

2.2 适用于分析海量小文件的自适应学习率函数

随机梯度算法(SGD)是一种基于梯度下降算法的优化方法，其通过随机选择样本来计算代价函数的梯度并更新模型参数。在 SVD++推荐算法中，一般使用 SGD 算法来训练模型，并使用学习率函数来控制模型参数更新的速率。这主要是因为：(1)SGD 基于每个样本的误差来更新模型参数，而不是基于整个数据集的误差，因此可以处理大规模数据集。(2)SVD++算法是一种基于矩阵分解的算法，它的目标是最小化评分预测误差的平方和。由于 SGD 可以在每个样本上进行更新，因此可以更快地收敛到最优解。(3)SVD++算法需要优化的参数非常多，包括用户和物品的隐含因子向量和偏置项。使用 SGD 可以高效地更新这些参数，而不会占用太多的计算资源。

学习率函数是指在训练过程中，学习率根据迭代次数或其他因素进行变化的函数。一般来说，学习率会随着迭代次数的增加而逐渐减小，以保证模型收敛到最优解。常见的学习率函数有固定学习率(Fixed Learning Rate)、常数衰减学习率(Constant Decay Learning Rate)、随机衰减学习率(Random Decay Learning Rate)、指数学习率(Exponential Learning Rate)、自适应学习率(Adaptive Learning Rate)等[6]函数。本文将提出一个新的自适应学习率函数以进一步应对海量大数据下背景下小文件合并的问题，提高收敛性能以及预测质量。

随机梯度下降法的更新公式可以简单表示为：

$$x(k+1) \leftarrow x(k) + \alpha * g(k) \qquad (2)$$

其中，α是学习率函数，g(k)是 x(k)的梯度，k 是迭代的次数。

指数学习率是目前常用的一种方法，且常被运用在 SVD++算法上，效果较好。其具体可表示为：

$$\alpha_{ELR}(k) = \alpha_0 * \alpha_1^k \qquad (3)$$

其中，k 是迭代的次数，α0、α1 是该函数的参数。由此公式，可以看出指数学习率衰减速度过快或过慢都会影响模型的性能。如果衰减速度过快，会导致模型在训练初期无法收敛；如果衰减速度过慢，则可能导致模型在训练后期无法有效更新，影响

训练效果。ELR 函数的问题就在于学习率下降速度过慢，在后期可能会导致收敛速度变慢，甚至陷入局部最小值。因此，针对指数学习率函数的一些局限性，本文提出了一种适用于 SVD++的自适应学习率函数：

$$\alpha_{ALR} = \frac{\alpha_0}{\alpha_1^k + \alpha_2 * \sqrt{k} + \alpha_3} \tag{4}$$

其中，k 是迭代的次数，α1、α2、α3 是该函数的参数。该函数在 ELR 函数的基础上增加了一个根号线性函数，并添加了一个常数项α3。这样做的目的是使学习率在早期具有更高的下降速度，以便更快地逼近全局最优解。其次，设计 a0 是为了将学习率的初始值控制在 a0 这个范围内，并且在训练过程中自适应地调整学习率。

SVD++的损失函数是用来度量模型在训练过程中预测值与真实值之间的差距，进而调整模型参数使得损失函数最小化。SVD++的损失函数 L 可以表示为：

$$\mathscr{L} = \sum_{(u,i)\in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2 + \frac{1}{|N(u)|} \sum_{j\in N(u)} \|y_j\|^2 \right) \tag{5}$$

其中，K 表示已知评分的样本集合，rui 表示实际评分，r^ui 表示预测评分，bu 表示用户 u 的偏置项，bi 表示小文件 i 的偏置项，qi 表示小文件 i 的隐含特征向量，pu 表示用户 u 的隐含特征向量，N(u)表示用户 u 访问过的小文件集合，N(u)表示用户 u 访问过的小文件数量，yj 表示小文件 j 的隐含特征向量，λ表示正则化参数，用于控制模型的复杂度。

采用随机梯度算法最小化目标函数的过程需要求每个变量的偏导数，即对目标函数对每个变量求偏导数，从而得到对应的梯度向量。然后根据梯度的方向来更新变量，使得目标函数不断逼近最优解。

2.3 基于并行分组的 SVD++算法

当需要处理大规模数据集时，常常会选用 Mapreduce。因为它是一个常用的并行计算框架，可以对数据进行分块处理并发地执行算法。然而，对于 SVD++算法来说，由于算法中需要进行矩阵的乘法和求逆等复杂的数学计算，很难将其直接应用到 MapReduce 中进行并行化处理。为了解决这个问题，可以使用 Scala 编写分布式计算程序以及使用 Spark 等分布式计算框架对 SVD++算法进行并行化处理。Spark 支持强大的分布式数据处理和计算能力，并且可以很方便地与 Scala 集成。通过在 Spark 中利用其支持的数据缓存机制和优化算法，可以大大提高 SVD++算法的处理效率和性能。本文

将用户访问记录以及小文件的相关信息的数据集表示为 Spark 的 RDD(弹性分布式数据集),使用 Scala 编写算法逻辑。通过适当的算法实现和优化，可在 Spark 集群上并行执行 SVD++算法。Spark 支持多种操作，如 Map、Reduce、Join 等，根据需要可以选择合适的操作对数据进行转换和计算。

具体的方法步骤为：

第一步，将小文件数据集加载到 Spark 的 RDD 中，每个小文件作为一个记录，其中包含文件名，用户 ID 以及从用户访问记录里读取到的相关信息。同时对向量进行初始化，包括包括用户和小文件的偏置(bi 和 bu)以及隐含的特征向量(pu 和 qi)

第二步，计算全局平均评分μ,通过计算得出训练集中所有评分记录的评分平均值。

第三步，计算参数值。计算每个用户和每个小文件的平均得分，即用户和小文件的偏置(bi 和 bu)。同时，为了计算用户的隐式反馈 yj,需要读取用户的历史访问记录，从中找到用户访问过的小文件集合 N(u),以便计算。

第四步，数据划分以及并行计算：将数据划分为多个数据块，确保每个数据块都可以独立处理。然后将数据块分布在不同的计算节点上，以便进行并行处理。最后可以使用 Spark 提供的高级 API(如 map、flatMap、reduceByKey 等)对数据块进行并行计算。

第五步，训练模型。采用上文中提到的自适应学习率函数以及随机梯度算法对 SVD++算法的模型参数进行更新[7]。具体步骤如下：

(1):开始迭代更新过程，可以设置迭代次数或收敛条件。

(2):每次迭代中，根据固定 q 或 p 的策略，按照下述步骤更新模型参数：

1):固定 q,更新 p:首先计算每个用户的 $|p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$

然后使用随机梯度下降算法更新每个用户的因子向量 p 和偏置项。

2):固定 p,更新 q、y 和小文件的偏置项：首先计算每个小文件的隐含因子向量 q 和邻域等级 y,以及小文件的偏置项。然后使用随机梯度下降算法更新每个商品的因子向量 q、邻域等级 y 和偏置项。

第六步，汇总和聚合模型参数。在每个计算节点上，汇总更新后的模型参数。使用 Spark 的聚合操作(如 reduce)将模型参数聚合到主节点上。

第七步：将聚合后的模型参数导出到适当的格式进行保存。

对于该算法的代码实现如下表 1 所示：

伪代码：基于并行分组的 SVD++算法
1 alpha0= initialAlpha0 2 alpha1= initialAlpha1 3 alpha2= initialAlpha2

```
4 alpha3= initialAlpha3#初始化自适应学习率函数参数
5 for iteration inrange(1,numIterations +1):#开始训练模型
6 for block in parallelDataBlocks: #并行处理每个数据块
7 for userId, fileData in block:
8 userVector= getUserVector(userId)
9 userBias= getUserBias(userId)
10 userBiasGradient=0.0
11 userVectorGradient=[0.0]* numFactors#更新用户因子向量和偏置项
12 for fileId in fileData.accessedFiles:
13 itemVector= getItemVector(fileId)
14 itemBias= getItemBias(fileId)
15 implicitFeedback= implicitFeedbacks[userId][fileId]#遍历用户访问的小文件集合
16 error= calculateError(userId, fileId)
17 userBiasGradient+= error + regularization * userBias
18 userVectorGradient=[userVectorGradient[i]+ error * itemVector[i]+ regularization *
userVector[i]for i inrange(numFactors)]#计算误差和梯度
19 itemBiasGradient= error + regularization * itemBias
20 itemVectorGradient=[error * userVector[i]+ regularization * itemVector[i]for i
inrange(numFactors)]#更新小文件的因子向量和偏置项
21 currentAlpha= alpha0 /(alpha1 ** iteration + alpha2 * iteration **0.5+ alpha3)#计算
自适应学习率
22 itemBiasGradient*= currentAlpha
23 itemVectorGradient=[itemVectorGradient[i]* currentAlpha for i
inrange(numFactors)]
24 updateItemParameters(fileId, itemBiasGradient, itemVectorGradient)#更新小文件
的因子向量和偏置项，乘以自适应学习率
25 userBiasGradient*= currentAlpha
26 userVectorGradient=[userVectorGradient[i]* currentAlpha for i
inrange(numFactors)]
27 updateUserParameters(userId, userBiasGradient, userVectorGradient)#更新用户的
因子向量和偏置项，乘以自适应学习率
28 aggregatedUserParameters= aggregateUserParameters(parallelDataBlocks)
29 aggregatedItemParameters= aggregateItemParameters(parallelDataBlocks)#汇总和
聚合模型参数
```

30 updateUserParameters(aggregatedUserParameters)
31 updateItemParameters(aggregatedItemParameters)#更新模型参数

3 与 K-means++算法结合

K-means++算法是一种经典的聚类算法，旨在将数据点划分为 K 个簇，使得每个数据点与所属簇的质心之间的距离最小化。在传统 K-means 算法中，初始质心的选择是随机的，容易导致陷入局部最优解。为了解决这一问题，K-means++算法在初始质心选择阶段引入了改进的方法，以选择初始质心，使得各个质心之间的距离较远，提高全局最优解的概率。具体步骤如下[8]:

1.从数据点集合中随机选择一个数据点作为第一个质心。

2.对于每个数据点，计算其到已选择质心集合中最近质心的距离 D(x, C),其中 x 为数据点，C 为已选择质心集合。

3.根据距离 D(x, C)与已选择质心集合中所有距离的加权概率，选择下一个质心，以保证各个质心的距离较远。

K-means++算法的初始质心选择策略能够使得聚类效果更加稳定和全局最优。

在本研究中，选择 K-means++算法作为聚类方法的主要原因在于其对初始质心选择的改进。由于小文件的数据分布可能不均匀，传统 K-means 算法的随机质心选择容易导致局部最优解，降低了聚类效果。而 K-means++算法能够更好地选择初始质心，提高全局最优解的概率，适应不同数据分布的情况。而本文将改进后的 SVD++算法得到的隐含特征向量作为 K-means++算法的输入数据。这样做的好处主要体现在以下几个方面：

1.考虑隐含关系：改进后的 SVD++算法能够在模型训练中考虑隐含反馈信息，对于评估小文件之间的相关性和相似性更为全面。将改进后的 SVD++算法得到的隐含特征向量作为 K-means++算法的输入数据，使得聚类过程不仅考虑了原始文件的内容特征，还考虑了文件之间的隐含关系。

2.综合多维信息：小文件的聚类不仅需要考虑文件内容特征，还需要考虑到文件之间的相关性。通过改进后的 SVD++算法得到的隐含特征向量，将小文件在隐含特征空间进行聚类，能够综合多维信息，更好地评估小文件之间的相似性。

3.统一特征空间：改进后的 SVD++算法将小文件映射到一个隐含特征空间中，这样在聚类过程中，数据点在相同的特征空间中进行距离计算，更好地比较小文件之间的相似性，避免了特征尺度不一致的问题。

具体的方法步骤如下：

1.数据预处理：对小文件进行预处理，包括数据清洗和特征提取。为了减少存储和计算的复杂性，我们选择提取文件内容的关键特征，例如小文件的大小、类型、创建时间等，将这些特征表示为向量形式。

2.改进 SVD++算法的应用：应用改进后的 SVD++算法对预处理后的小文件特征进行建模和训练。改进后的 SVD++算法通过隐含特征空间的分解，得到每个小文件在隐含特征空间中的表示，得到一组隐含特征向量。这些隐含特征向量捕捉了文件之间的隐含关系和相似性。

3.数据的准备：将 SVD++算法得到的隐含特征向量作为 K-means++算法的输入数据，将每个小文件的隐含特征向量视为一个数据点。将小文件在隐含特征空间中表示，使得在聚类过程中能够综合考虑文件内容特征和隐含关系。

4.初始质心选择：在 K-means++算法的初始质心选择阶段，通过改进的方法选择初始质心，以避免陷入局部最优解。具体步骤如下：

(1)随机选择一个数据点作为第一个质心。

(2)对于每个数据点，计算其到已选择质心集合中最近质心的距离 D(x, C),其中 x 为数据点，C 为已选择质心集合。

(3)根据距离 D(x, C)与已选择质心集合中所有距离的加权概率，选择下一个质心，以保证各个质心的距离较远。

5.聚类迭代：在得到初始质心后，使用 K-means++算法进行迭代更新质心和重新分配数据点到最近的质心的步骤。迭代过程中，每个数据点将被分配到与其最近的质心所代表的簇中。

6.小文件合并：在 K-means++算法完成聚类后，根据聚类结果将属于同一簇的小文件进行合并。这样，相似的数据块被聚集在一起，优化了数据的存储方式，减少了存储冗余，提高了存储空间利用率。具体的伪代码如下表 2 所示。

伪代码：K-means++算法的应用
1 defpreprocess_and_apply_svdplusplus(file_set): 2 # 对小文件进行预处理，得到特征向量集合 features 3 features= preprocess(file_set) 4 # 使用改进的 SVD++算法对 features 进行训练，得到每个小文件在隐含特征空间的表示 latent_vectors 5 latent_vectors= svdplusplus_train(features) 6 return latent_vectors 7 # K-means++算法的聚类过程和小文件合并 8 defkmeansplusplus_and_file_merge(latent_vectors, K):

```
9 # 数据准备
10 data_set= latent_vectors
11 # 初始质心选择(改进的 K-means++算法)
12 centroids= kmeansplusplus_initialization(data_set, K)
13
14 # 聚类迭代
15 max_iterations=100
16 iterations=0
17 converge=False
18 whilenot converge and iterations < max_iterations:
19 # 分配数据点到最近的质心，得到聚类结果 cluster_result
20 cluster_result= assign_data_points_to_clusters(data_set, centroids)
21 # 计算簇内所有数据点的平均向量 new_centroid
22 new_centroids= calculate_new_centroids(cluster_result)
23 # 判断是否收敛
24 if has_converged(centroids, new_centroids):
25 converge=True
26 else:
27 centroids= new_centroids
28 iterations+=1
29 # 小文件合并
30 merged_files= merge_files_in_clusters(cluster_result)
31 return merged_files
```

4 实验结果分析

为了验证本文提出的改进 SVD++算法与 K-means++算法结合的小文件合并方案在 Hadoop 中的应用是否有效地提高了存储小文件的效率，本节将对实验结果进行对比分析。我们将分别比较三种方案：HAR 方案、原始合并方案和本文提出的合并方案，在名字节点的内存消耗、单位文件读取时间和单位文件上传时间方面的表现。为了确保实验的准确性，每组实验都会进行七次测试。每组测试环境的 HDFS 集群均由一个 NameNode 和五个 DataNode 组成。操作系统均为 64 位 CentOS8.3,Hadoop 版本为 3.1.3,JDK 版本为 OpenJDK11。

4.1 内存消耗对比

我们首先比较三种方案在名字节点的内存消耗方面的差异。实验使用 10000 个不同大小的小文件作为测试数据集，分别应用 HAR 方案、原始合并方案和本文提出的合并方案进行文件合并操作，记录每种方案下名字节点的内存消耗情况。结果如图 1 所示：

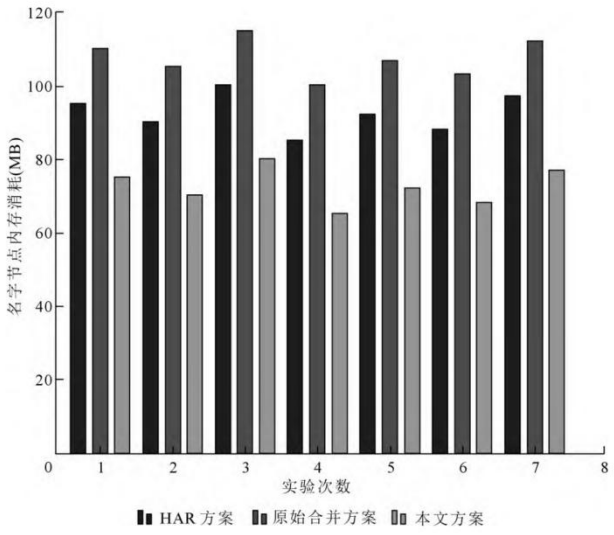

图 1　不同方案的内存消耗对比图

从图 1 中可以观察到，本文提出的合并方案在名字节点的内存消耗方面表现明显优于 HAR 方案和原始合并方案。在 HAR 方案中，由于每个小文件都会生成一个索引文件，导致存储空间的浪费和内存占用的增加。而原始合并方案在处理大量小文件时，由于合并后的较大文件可能存在较多的空洞，也会增加内存消耗。相比之下，本文的合并方案通过使用改进的 SVD++算法和 K-means++算法，将小文件合并成较少的聚类簇，降低了内存占用，从而有效地减少了名字节点的内存消耗。

4.2 单位文件读取时间对比

在实验中，我们使用相同的测试数据集，分别采用 **HAR** 方案、原始合并方案和本文提出的合并方案对小文件进行合并，并记录读取单个文件所需的平均时间。结果如图 **2** 所示：

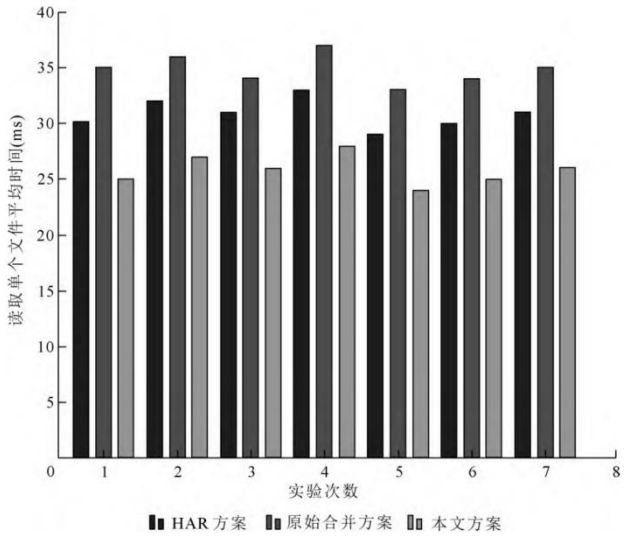

图 2　不同方案的单位文件读取时间对比图

从图 **2** 中可以看出，本文提出的合并方案显著降低了单位文件的读取时间。在 **HAR** 方案中，由于每个小文件都需要通过索引进行查找，读取时间相对较长。而原始合并方案虽然减少了索引查找，但在较大的合并文件中读取目标文件仍需要较长时间。相比之下，本文的合并方案根据聚类结果将文件合并成较少的簇，每个簇中的文件具有相似的特征，因此在读取时减少了查找和定位的时间，从而显著降低了单位文件的读取时间。

4.3 单位文件上传时间对比

实验使用相同的测试数据集，分别采用 **HAR** 方案、原始合并方案和本文提出的合并方案对小文件进行合并，并记录上传单个文件所需的平均时间。结果如图 **3** 所示：

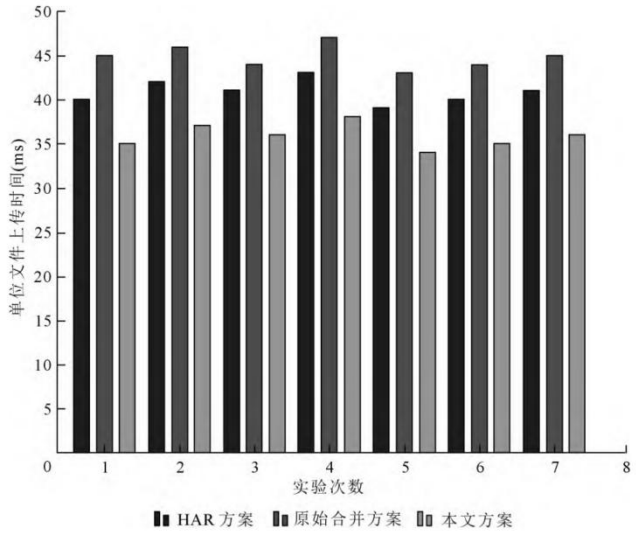

图 3　不同方案的单位文件上传时间对比图

4.4 综合分析

综合上述实验结果对比，本文提出的改进 **SVD**++算法与 **K-means**++算法结合的小文件合并方案在名字节点的内存消耗、单位文件读取时间和单位文件上传时间三个性能指标上均优于传统的 **HAR** 方案和原始合并方案。通过降低内存负载、提升文件读取性能和文件上传性能，本文的方案有效地改进了 **Hadoop** 存储小文件的效率，对于海量小文件的存储和处理具有重要的实际意义。

5 结语

本文针对 **Hadoop** 存储小文件效率低的问题，提出了一种改进 **SVD**++算法与 **K-means**++算法结合的小文件合并方案。通过对比实验，我们验证了该方案在名字节点的内存消耗、单位文件读取时间和单位文件上传时间方面的优越性。本文的合并方案将小文件聚类合并，降低了存储空间的消耗，提高了文件的读取和上传性能，有效地改善了 **Hadoop** 处理海量小文件的性能。该方案为 **Hadoop** 系统的小文件处理提供了一种高效可行的解决方案。

参考文献

[1]付红阁，姜华，张怀锋.基于 Hadoop 的海量统计小文件存取优化方案[J].聊城大学学报(自然科学版),2016,29(01):102-106.

[2]朱莉；基于 HDFS 的云存储小文件合并优化方法研究[J/OL].船舶职业教育，2023(02 vo 11):62-65.

[3]A M A A,A R B.Dynamic Merging based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop[J].Procedia Computer Science,2018,132:1626-1635.

[4]PENG J feng,WEI W guo,ZHAO H min,等.Hadoop Massive Small File Merging Technology Based on Visiting Hot-Spot and Associated File Optimization[M/OL]//REN J,HUSSAIN A,ZHENG J,等.Advances in Brain Inspired Cognitive Systems:卷10989.Cham:Springer International Publishing,2018:517-524[2023-07-23].

[5]王燕，李凤莲，张雪英，等.改进学习率的一种高效SVD++算法[J].现代电子技术，2018,41(3):6.

[6]JIAO J,ZHANG X,LI F,等.A Novel Learning Rate Function and Its Application on the SVD++ Recommendation Algorithm[J/OL].IEEE Access,2020,8:14112-14122.

[7]CAO J,HU H,LUO T,等.Distributed Design and Implementation of SVD++ Algorithm for E-commerce Personalized Recommender System[M/OL]//ZHANG X,WU Z,SHA X.Embedded System Technology:卷572.Singapore:Springer Singapore,2015:30-44[2023-05-08].

[8]CHANDGUDE P,BHAGWAT A,AUTADE M,等.A Novel approach for k-means++ approximation using Hadoop[J].