文章编号:1007-5321(2020)02-0116-06 DOI:10.13190/j.jbupt.2019-092

Spark 环境下基于数据倾斜模型的 Shuffle 分区优化方案

阎逸飞, 王智立, 邱雪松, 王嘉潞
(北京邮电大学 网络与交换技术国家重点实验室, 北京 100876)

摘要: 针对 Spark 分布式平台在 shuffle 阶段中导致数据量分配不均衡的问题,首先分析了 Spark 平台中数据倾斜的原因,建立了一个可以统一量化 shuffle 后 key-value 数据倾斜程度的倾斜模型;基于倾斜模型提出了一个可以解决 Spark 平台中多种数据倾斜问题的 shuffle 分区方案. 该分区方案首先对 Map 阶段的输出数据进行采样,预测出全局中间数据的大小,再根据基于哈希的最佳适应算法对采样数据进行预分区,得到一张预分区表,最后根据预分区表对全部的中间数据完成分区. 在 key 和 value 这 2 种不同倾斜情况下的实验结果表明,该 shuffle 分区方案具有普适性和高效性,可以有效处理 key 和 value 倾斜的情况.

关 键 词: 数据倾斜; Spark; shuffle; 分区算法; 负载均衡
中图分类号: TP399 **文献标志码**: A

A Shuffle Partition Optimization Scheme Based on Data Skew Model in Spark

YAN Yi-fei, WANG Zhi-li, QIU Xue-song, WANG Jia-lu
(State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: For the problem of uneven distribution of data caused during the shuffle phase in the Spark distributed platform, the reason of Spark's low efficiency in processing skewed data is analyzed, then a skew model that can uniformly quantize the skew degree of key-value data after shuffle is proposed. Based on this skew model is established, and a shuffle partitioning scheme that can solve various data skew problems in the Spark platform is proposed. Firstly, the output data of the Map stage is sampled, the size of the intermediate data is predicted, and then the sampled data is pre-partitioned according to the Hash-based best fit algorithm. Finally, all the intermediate data is partitioned according to the pre-partition situation. In the cases of key skew and value skew, the experimental results show that this shuffle partitioning scheme is universal and efficient, and can effectively handle the situation of key and value skew.

Key words: data skew; Spark; shuffle; partitioning algorithm; load balancing

Spark 是一种当前非常流行的基于内存计算的分布式计算平台,具有简单、通用、高效等特点. 由于 Spark 具有内存计算的特性,会把中间数据保存在内存中,降低磁盘 I/O 负载,提高数据迭代计算的效率. 数据倾斜是分布式计算中经常会出现的问题. Spark 框架本身没有针对数据倾斜问题的有效

收稿日期: 2019-05-28
作者简介: 阎逸飞(1993—), 男, 硕士生.
通信作者: 王智立(1975—), 男, 副教授, E-mail: zlwang@bupt.edu.cn.

应对措施，Spark 在 shuffle 阶段，会使后续任务的数据集的分布不均、出现大量的数据拉取等问题，影响后续任务的执行效率，增加计算时间. 因此，对 Spark 的 shuffle 机制的优化是一个非常值得研究的课题.

1　相关研究

数据倾斜是分布式数据处理中经常会遇到的问题. Yu 等[1]提出的 Spark adaptive skew mitigation 自适应策略，可以通过数据迁移，缓解数据倾斜的问题. Liu 等[2]提出的 SP-Partitioner 是一种更适用于计算流数据情况下解决数据倾斜问题的分区器，但是抽样方法采用了间隔抽样，在数据巨大的情况，分区表会变的大的难以维护. Tang 等[3]提出了一种对于中间数据的 splitting and combination 算法，在对输入数据进行抽样吼，根据此算法得到相应的预测分区表. 之后再开始正式执行作业任务，到 shuffle 阶段根据上一步得到的预测分区表来进行分区，使下一阶段任务的输入数据更加平均.

2　Spark 的 shuffle 机制

Spark 的 shuffle[4]过程是将 Map 的输出数据重新分区，然后作为各个 Reduce 任务的输入数据. Spark 中的 shuffle 大体分为了 shuffle write 和 shuffle fetch 两个阶段. 过程如图 1 所示，把 Map 端对数据的分区和存储称为 shuffle write，将 Reduce 端拉取数据的过程称为 shuffle fetch. Spark 的中间数据总是以 key-value 这样的键值对形式存在的.

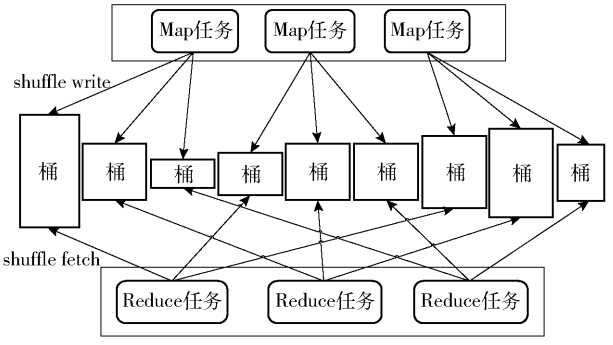

图 1　spark shuffle 的过程

每个 Map 任务会根据 Reduce 任务的数量创建相应的桶，shuffle 过程中桶的数量为 MR（M 为 Map 任务的数量，R 为 Reduce 任务的数量），图 1 中有 3 个 Map 任务，3 个 Reduce 任务，因此共有 9 个桶. 在 shuffle write 的过程中，会把 Map 任务得到的所有中间数据根据一定规则分区放置在各个桶中. 在 Spark 中，默认的分区方法是 Hash 分区，即通过把 key 的哈希值对 R 取余，就可以得到该数据所需要存放的数据桶的 id. 最后当 Reduce 任务启动后，会根据所需要的数据，拉取不同桶中的数据，然后对数据进行 Reduce 操作，计算得到最后的结果.

由于 Hash 分区的方法会把具有相同 key 的中间数据分配到同一个桶中. 当某些具有相同 key 的中间数据组成数据簇的大小很大时，将导致数据倾斜，则该数据簇对应的 Reduce 任务的执行时间将会很长，甚至出现内存溢出的情况.

由于中间数据都是 < key，value > 元组，所以数据倾斜有两方面的原因.

1）key 倾斜：当某一种 key 元组的数量很多时，会使数据簇的大小变大.

2）value 倾斜：当 value 为可变大小时，某一种 key 所对应 value 的大小很大，而其他 key 所对应的 value 很小时，也会造成数据的倾斜.

在以往的研究中，大家往往只关注到 key 的倾斜而忽略了 value 的倾斜. 因为 Reduce 任务对每一个数据簇进行统一的处理正是对 value 进行计算和处理，所以最终处理的时间由 < key，value > 所占空间的大小决定[5]，因此需要对这 2 种倾斜情况进行统一建模处理.

3　Key-Value 数据倾斜模型

Spark 的中间数据用 $I \subseteq K \times V$ 来表示，K 和 V 分别是 key 和 value 的集合. 假设中间数据共有 m 个不同的 key，key 的集合可表示为

$$K = \{k_1, k_2, \cdots, k_m\} \tag{1}$$

同样，value 的集合表示为

$$V = \begin{Bmatrix} v_1^{k_1}, v_2^{k_1}, \cdots, v_{n_{k_1}}^{k_1}, \\ \cdots, v_j^{k_i}, \cdots, \\ v_1^{k_m}, v_2^{k_m}, \cdots, v_{n_{k_m}}^{k_m} \end{Bmatrix} \tag{2}$$

其中 $v_i^{k_i}(1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n_{k_i})$ 是 k_i 的一个 value，n_{k_i} 是中间数据中以 k_i 为 key 元组的数量. 因此，所有中间数据 I 就可以用式(3)表示为

$$I = \begin{Bmatrix} (k_1, v_1^{k_1}), (k_1, v_2^{k_1}), \cdots, (k_1, v_{n_{k_1}}^{k_1}), \\ \cdots, (k_i, v_j^{k_i}), \cdots \\ (k_m, v_1^{k_m}), (k_m, v_2^{k_m}), \cdots, (k_m, v_{n_{k_m}}^{k_m}) \end{Bmatrix} \tag{3}$$

I 中的每一个数据代表中间数据的一个 key-value 元组,一个数据簇是具有同一个 key 的所有元组的集合,可被表示为

$$C_{k_i}=\{(k_i,v_1^{k_i}),(k_i,v_2^{k_i}),\cdots,(k_i,v_m^{k_i})\} \tag{4}$$

式(4)是 key 等于 k_i 时的数据簇.

中间数据被分配到不同的桶中,被放置在一个桶 j 中的 key 可以表示为

$$\{k_x^j \mid k_1^j,\cdots,k_{l_j}^j \in K;\ 1\leqslant j\leqslant N;1\leqslant x\leqslant l_j\} \tag{5}$$

这里的 l_j 是 j 桶中 key 的数量,$1\leqslant l_j\leqslant m$. 假设共有 N 个数据桶,一个桶会放置 1 个或多个数据簇,因此,桶 j 中可放置的数据簇表示为

$$V_j=\{C_{k_1^j},C_{k_2^j},\cdots,C_{k_{l_i}^j}\},1\leqslant j\leqslant N \tag{6}$$

$S(v_i^{k_i})$ 表示 $v_i^{k_i}$ 所占空间的大小,对于 k_i 数据簇的总大小为

$$Q_{k_i}=\sum_{i=1}^{n_{k_i}} S(v_i^{k_i}) \tag{7}$$

n_{k_i} 是 C_{k_i} 数据簇中元组的数量. 每个元组的平均大小为

$$\overline{Q}_{k_i}=\frac{Q_{k_i}}{n_{k_i}} \tag{8}$$

每一个桶的总大小为

$$O_j=\sum_{i=1}^{l_i} n_{k_i^j}\overline{Q}_{k_l} \tag{9}$$

其中:l_i 为桶 j 中的数据簇总数,$n_{k_i^j}$ 为数据簇 $C_{k_i^j}$ 中元组的数量. 每个桶的平均容量为

$$B_{\text{mean}}=\frac{\sum_{i=1}^{m} Q_{k_i}}{N} \tag{10}$$

在数据统计中,变异系数是一个计算数据离散程度的标准方法. 因此,使用变异系数来计算中间数据分区之后各个数据桶数据量的数据倾斜度(DoS,data of skewness)为

$$D=\frac{\sqrt{\sum_{j=1}^{N}(O_j-B_{\text{mean}})^2}}{B_{\text{mean}}} \tag{11}$$

当 D 越小时,表示各个桶所存储的数据量越近似,数据倾斜度越小;反之表示各个桶的数据量不均衡. 当 D 等于 0 时,表示各个桶的数据量完全相同.

4 shuffle 优化方案

为了解决 shuffle 过程中的数据倾斜问题,基于前一步建立的统一 key 和 value 倾斜的数据倾斜模型,提出一个可以解决 shuffle 过程中各种数据倾斜问题的 shuffle 分区方案. 分区方案有以下 3 步:

1) 数据分区与预测;

2) 制定数据分区策略;

3) 完成数据分区.

4.1 数据分区与预测

先使用水塘抽样法对分布在集群中各个工作机上的中间数据进行高效等概率抽样,只需要对中间数据扫描一遍即可完成抽样,并且除去样本空间,不需要占用额外的空间,如算法 1 所示.

算法 1 水塘抽样法

输入:抽样容量 m;中间数据 I

输出:样本数据 sample;中间数据总个数 n;总空间 S

```
1  sample = [ ];
2  n = 0;
3  S = 0;
4  for each ⟨k_i, v_j⟩ in I do
5     vSize = space (v_j); //计算每个 value 所占的空间
6     S = S + vSize; // 累加 value 所占的总空间
7     n = n + 1;
8     if (n≤m) then
9        sample = sample ∪ ⟨k_i, vSize⟩;
10    else
11       r = rand(0, n); //随机 0 - n 之间的整数 r
12          if (r < m) then
13                sample[r] = i;
14             end if
15    end if
16 end for
```

完成数据采样之后可以得到样本数据、中间数据总个数和总空间信息. 借助这些数据信息和抽样率可以近似描绘出全部中间数据的特征. 先把样本数据中所有具有相同 key 的数据汇总,并对其大小进行加和,再根据抽样率计算出全量数据中每个数据簇的大小和其中的元组个数. 这些数据可表示为

$$M=\{\langle k_1,\overline{S}_1,S_1\rangle,\cdots,\langle k_m,\overline{S}_m,S_m\rangle\} \tag{12}$$

其中:k_i 表示一种 key,$\overline{S}_i$ 表示 k_i 代表的数据簇中每个元组的平均空间,S_m 表示 k_i 代表的数据簇所有元组的总空间.

另外,通过抽样得到中间数据占的总空间 S 和用户定义的分区个数 N,可以计算出每个桶中理想的容量.

$$B_{\text{mean}} = S/N \tag{13}$$

4.2 制定数据分区策略

根据上一步预测得到的相关数据，这一步要以相对最优的方式把数据簇放置在合理的桶中，这个问题类似一个装箱问题. 装箱问题已经被证明是一个 NP-Hard 问题，无法在多项式时间内找到最优解，只能尽可能在较短的时间内找到近似最优解，于是设计了预分区算法，如算法 2 所示.

算法 2　基于哈希的最佳适应算法

输入：数据簇信息 M；桶的平均容量 B_{mean}；数据桶数量 N

输出：预分区信息集合 P

```
1  RB = [ ]; //记录每个桶的剩余容量
2  P = { }; //预分区信息集合
3  for k←1 to N do//初始化每个数据桶的剩余容量
4    RB_k = B_mean
5  end for
6  for each ⟨k_i, S̄_l, S_i⟩ in M do
7    x = hash(k_i) % N//用哈希法得到初始目标桶
8    if (RB_x ≥ S_i) then
9        y = x
10   eles if (∃RB ≥ S_i) then
11       y = 满足 S_i 大小且剩余容量最小的桶 ID
12   else
13       y = 剩余容量最大的桶 ID
14   end if
15   P = P ∪ {⟨k_i, y, S_i/S̄⟩}
16   RB_y = RB_y − S_i
17 end for
```

算法的输入参数数据簇信息 M 和桶的平均容量 B_{mean} 分别在式(12)和式(13)中已进行说明，算法的输出结果 P 表示预分区集合，其中包含 key 的信息和该 key 应该被分配到对应桶的 id 以及相应的分配个数.

算法 2 的思路是先用 B_{mean} 初始化每个桶的剩余容量，再用 Hash 的方法得一个最初的目标桶. 若该桶已满，则从剩余的数据桶中找到能放下该数据簇桶中剩余容量最小的桶；若所有桶的剩余容量都小于数据簇的大小，则找剩余容量最大的桶放置该数据簇. 本算法借助了解决装箱问题的最佳适应算法，另外又加上了哈希分区法快速的优势. 可以在 $O(\text{nlogn})$ 的时间复杂度下完成数据分区，这里的 n 是数据簇的个数，一般情况下数据簇个数会比中间数据小很多个数量级. 从理论上分析，该算法有非常好的执行性能和良好的预分区结果. 该算法最终输出一张预分区表，表中记录每个 key 值对应的数据桶以及其要放置在桶中的数量.

4.3 完成数据分区

这一步将会根据上一步得到的预分区表对全量的中间数据完成分区. 首先在表中查询是否有当前中间数据的 key，如果有则按照分区表中指示的桶进行放置；若不在预分区表中，则根据 Hash 分区的方法进行分区. 在实现中预分区表用 HashMap 来保存，对每个数据分区的过程仅需要 $O(1)$ 的时间复杂度，分区过程十分高效. 通过这样的分区方案可以使 shuffle 之后的各个 reduce 任务接收到的数据量近似相同.

5 实验验证

实验环境为 7 台虚拟机搭建起来的 Spark 和 HDFS 集群，每个节点的硬件及配置情况如表 1 所示.

表 1　集群配置表

节点类型	数量	CPU	内存/GB	硬盘/GB	部署模式	系统环境
Master	1	2 核，2.6 GHz	6	200	standalone	Ubuntu16.04，JDK1.8 Hadoop2.7.5，Spark2.2.0
Slave	6	2 核，2.6 GHz	6	200		

为了方便设置原始数据的倾斜度，在实验中采用 zipf 定律[6]生成 key 和 value 倾斜度不同的原始数据，用于模拟真实的倾斜数据. 选取在倾斜数据下 Spark 作业的执行时间和 shuffle 之后的数据均衡程度 DoS 作为 shuffle 机制的性能衡量标准. 为了验证 shuffle 优化方案的性能，选取了 2 个对照分区实验.

1）Hash 分区. 这是 Spark 平台默认使用的方法. 该算法较为简单，直接使用中间数据的 key 哈希值对分区个数取模，得到该 key 的分区 id.

2）Range 分区. 这是 Spark 中提供的另一个分区方法，其作用是把一定范围内的数据映射到一个

分区中,并尽力保持每个分区中的数据个数比较均匀. 该机制中也需要先对中间数据进行抽样.

5.1 key 倾斜实验

通过 word count[7] 作业来验证 key 倾斜情况下该分区方案的性能,首先用 zipf 方法生成 key 倾斜度为 0.3、0.7、1.1 的 3 组数据,每个文本数据的大小为 3 GB.

由于 shuffle 优化方案 DPA(dynamic partition algorithm)涉及抽样,先用不同的抽样率计算各个作业的执行时间. 图 2 所示为抽样率为 5%、10%、15%和 20%情况下 DPA 的作业执行时间. 每个抽样率在不同倾斜度下的执行效率,为了避免由抽样率不同而造成的影响,选取这 4 种抽样率结果的平均值来计算 DPA 的执行时间和 DoS.

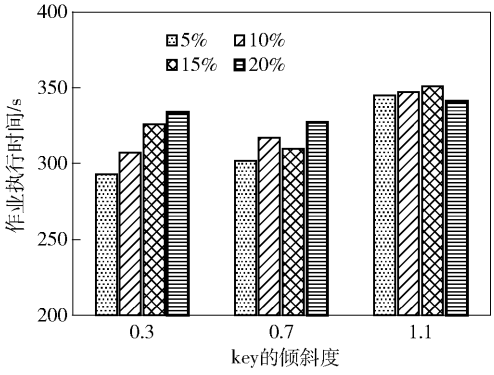

图 2 DPA 在不同抽样率下的执行时间

图 3 为 Hash、DPA 和 Range 3 种分区方案的在 0.3、0.7 和 1.1 倾斜度下的 DoS 的变化情况. 在倾斜度为 0.3 时,它们的 DoS 较为相近,差异不大. 当倾斜度为 0.7 以上时,Hash 方法的 DoS 增长非常快,Range 方法的 DoS 一直比较平稳的增长,DPA 方案具有非常好的负载均衡能力,DoS 一直维持在一个非常低的水平.

图 4 所示为 3 种分区方案的作业执行时间对比. 当 key 的倾斜度为 0.3 时,Hash 的方案速度最快,因为在分区时不需要花费任何额外的时间,可以较好的处理很低倾斜度的数据. 但随着倾斜度的升高 Hash 的执行时间快速升高,在倾斜度 1.1 时作业执行时间最长. DPA 方案执行时间在 0.7 倾斜度以上一直是最低的,而且执行时间非常稳定. Range 虽然执行时间较稳定,但是执行时间较长. 从实验结果可以看出 DPA 可以有效应对 key 倾斜的情况,具有很好的负载均衡能力和高效的执行效率.

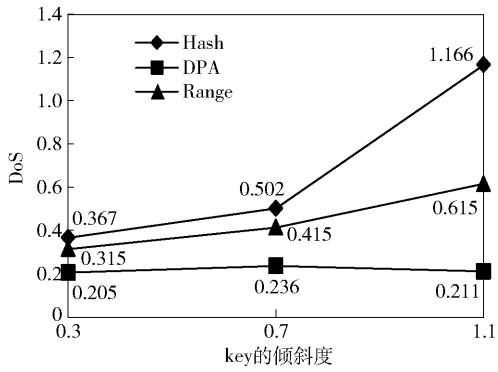

图 3 3 种分区方案的 DoS

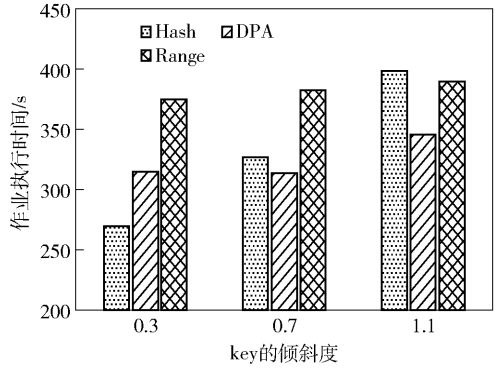

图 4 3 种分区方案的执行时间

5.2 value 倾斜实验

通过 search value[8] 作业来验证在 value 的倾斜情况下 DPA 方案的性能. 同样,用 zipf 方法生成 value 倾斜度为 0.3、0.7、1.1 的 3 组数据. key 的倾斜度保持在 0,即 key 随机分布,每个文本数据的大小为 3 GB.

图 5 为 value 倾斜度升高的情况下,DPA 方案在不同抽样率下的执行时间. 可以看出,在该数据集的情况下,20%的抽样率执行时间一直较长,但这并不能表示 20%的抽样率不适合于其他数据集. 与

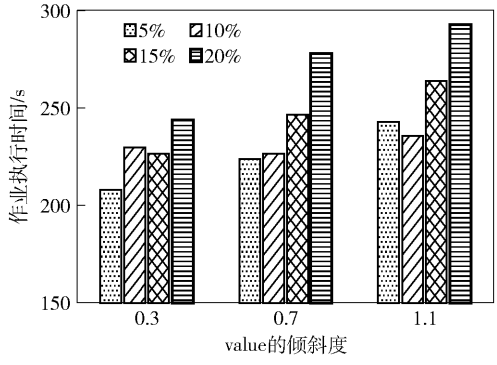

图 5 DPA 在不同抽样率下的执行时间

前一个实验相同，选取这 4 种抽样率结果的平均值来计算 DPA 的执行时间和 DoS.

图 6 为 value 倾斜情况下 3 种方案 DoS 的变化情况. 随着倾斜度的升高，Hash 和 Range 方案的倾斜程度快速升高，但 DPA 的 DoS 一直较小，可以保持分区后各个桶中数据量的平衡.

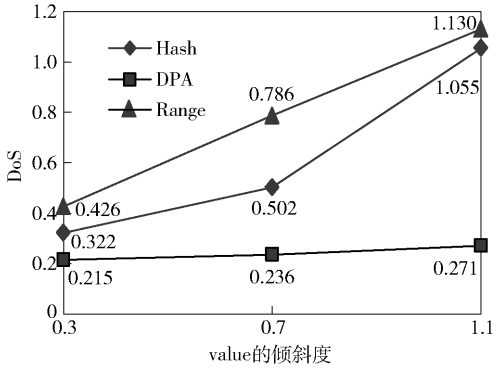

图 6　3 种分区方案的 DoS

图 7 为 3 种方案的执行时间，Range 方法在 3 种不同的倾斜度下的执行时间一直是最长的，Hash 方法在 0.3 这种很低倾斜度下的执行时间最短，在其他倾斜情况下 DPA 的执行时间最短，而且执行时间很稳定. 从实验结果可以看出，DPA 可以有效应对 value 倾斜的情况，具有良好的负载均衡能力和高效的执行效率，符合方案设计的预期.

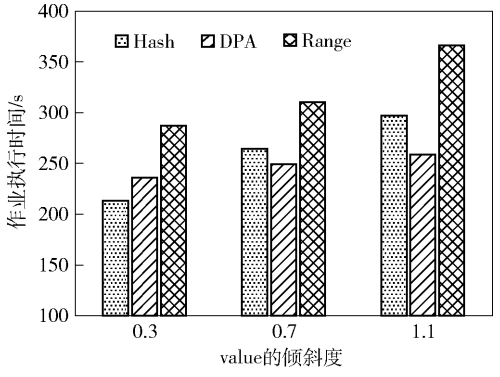

图 7　3 种分区方案的执行时间

6　结束语

针对 Spark 中 shuffle 阶段导致数据倾斜的问题，提出了一个可以统一量化 shuffle 后 key-value 数据倾斜程度的倾斜模型，再基于此倾斜模型提出了一个新的 shuffle 分区方案. 最后通过验证实验可以看出该模型和方案具有普适性和高效性，可以有效处理 key 倾斜的情况和 value 倾斜的情况.

参考文献：

[1] Ibrahim S, Jin H, Lu L, et al. Leen: locality/fairness-aware key partitioning for mapreduce in the cloud[C]// 2010 IEEE Second International Conference on Cloud Computing Technology and Science. [S. l.]: IEEE, 2010: 17-24.

[2] Liu G, Zhu X, Wang J, et al. SP-partitioner: a novel partition method to handle intermediate data skew in spark streaming[J]. Future Generation Computer Systems, 2018, 86: 1054-1063.

[3] 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.

[4] Davidson A, Or A. Optimizing shuffle performance in spark[EB/OL]. 2013[2018-11-25]. https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F13/projects/reports/project16_report.pdf.

[5] Gufler B, Augsten N, Reiser A, et al. Handling data skew in MapReduce[J]. Closer, 2011, 11: 574-583.

[6] De La Vega W F, Lueker G S. Bin packing can be solved within $1+\varepsilon$ in linear time[J]. Combinatorica, 1981, 1(4): 349-355.

[7] Adamic L A, Huberman B A. Zipf's law and the internet[J]. Glottometrics, 2002, 3(1): 143-150.

[8] Smith T, Simmons R. Heuristic search value iteration for POMDPs[C]// Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence. [S. l.]: AUAI Press, 2004: 520-527.