计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(12):299−307 [doi: 10.15888/j.cnki.csa.008224]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于节点负载的数据动态分区①

孟令伍[1], 杨阳朝[2], 黄晓明[2], 练丽萍[3]

[1](南京莱斯网信技术研究院有限公司, 南京 210014)
[2](深圳市网联安瑞网络科技有限公司, 深圳 518038)
[3](中电科新型智慧城市研究院有限公司, 深圳 518026)
通讯作者: 孟令伍, E-mail: jsjkx4@163.com

摘　要: 一种基于节点负载的数据动态分区系统, 主要考虑节点 CPU、内存、带宽负载情况, 首先采用二次平滑法预测节点的负载, 再结合 AHP 和熵值指标权重法得到每个节点的处理能力, 最后针对不同应用场景动态地调整系统的负载均衡性, 提高应用的响应速度; 该系统主要包括负载监测采集、预测、数据预分区、数据迁移等模块. 由于分布式环境存在节点资源的异构性, 为了数据分析计算过程中减少节点之间数据的传输, 充分利用节点计算资源, 通过负载均衡性提高应用分析的并行计算速度. 为此, 本文提出一种基于节点负载的数据动态分区机制和策略来改善系统负载均衡性及提高应用的响应速度, 辅助相关工作人员完成决策. 本论文结合 Spark 和 Elasticsearch 集成的数据分析应用场景进行测试.

关键词: 负载均衡; 动态分区机制; Spark; Elasticsearch

引用格式: 孟令伍,杨阳朝,黄晓明,练丽萍.基于节点负载的数据动态分区.计算机系统应用,2021,30(12):299–307. http://www.c-s-a.org.cn/1003-3254/8224.html

Dynamic Data Partition Based on Node Load

MENG Ling-Wu[1], YANG Yang-Chao[2], HUANG Xiao-Ming[2], LIAN Li-Ping[3]

[1](Nanjing Rice Net Information Technology Research Institute Co. Ltd., Nanjing 210014, China)
[2](Shenzhen CyberArey Network Technology Co. Ltd., Shenzhen 518038, China)
[3](CETC New Smart City Research Institute Co. Ltd., Shenzhen 518026, China)

Abstract: A dynamic data partition system based on node load mainly considers the load of CPU, memory, and bandwidth of nodes. It first uses the quadratic smoothing method to predict the load of nodes, then combines AHP and entropy index weight to get the processing capacity of each node, and finally dynamically adjusts the load balance of the system for different application scenarios to improve the response speed of applications. It includes the modules of load monitoring and collection, prediction, data pre-partition, and data migration. Given the heterogeneity of node resources in a distributed environment, it aims to reduce the data transmission between nodes in the process of data analysis and calculation, make full use of node computing resources, and improve the parallel computing speed of application analysis through load balancing. Therefore, this study proposes a dynamic partition data mechanism based on node load to improve the system load balance and application response speed and assist the relevant staff in making the decision. This study combines data analysis application scenarios integrating Spark and Elasticsearch for testing.

Key words: load balancing; dynamic partition mechanism; Spark; Elasticsearch

① 收稿时间: 2021-03-02; 修改时间: 2021-03-29; 采用时间: 2021-04-13

1 引言

大数据[1] 的发展, 带动了各种分布式计算、存储框架的发展. 其中 Spark、Flink 等作为开源主流的分布式计算框架, HBase、MemSQL、Elasticsearch 等作为开源主流的分布式存储框架. 但是计算存储框架都存在分区不合理引发数据倾斜[2,3] 而导致的集群负载不均衡现象, 大大降低了应用的分析性能. 因此, 为了提高集群数据分析性能, 有必要对数据分区策略进行研究, 最终提高数据分析的响应速度, 快速为企业提供决策, 增加效益.

分布式存储框架 Elasticsearch[4] 采用主从结构, 使用路由 Hash[5] 作为存储方式, 以数据分区 Shards 作为物理分区, 底层依托 Lucene 倒排索引结构, 并且支持文本分词, 非常适合关键词搜索分析. Spark 同样采用主从结构, Master 节点 (主节点) 管理整个集群的资源, Worker 节点 (从节点) 管理各计算节点的资源, 定期向 Master 节点汇报节点资源情况, 并启动 Executor 进行计算.

Spark-Elasticsearch[6] 集成的应用场景, 采用 local 方式读取数据分析的方式, 通过 Elasticsearch Spark Connect 组件将二者集成, Spark 中 Master 与 Elasticsearch 中主节点连接起来, 然后 Spark 的 Worker 节点可以通过 Master 节点获取到 Elasticsearch 中主节点的元数据信息, 其元数据信息包括数据有哪些分区及各分区存储在哪些节点上, 从而实现程序进行数据分析过程中, Spark 的 Worker 节点利用 esRDD 接口本地化并行地从 Elasticsearch 存储节点进行数据读写、计算分析, 如图 1 所示. Elasticsearch 中最小物理分区是 Shard, 目前每个节点默认分配相同的 Shard 数, 这会因为集群节点资源的异构性[7] 导致节点处理能力的不同而引发节点之间数据倾斜问题. 由于此框架中 Spark 采用 local 分析数据的方式, 即数据在哪个节点上面, 就在相应节点上进行分析处理, Elasticsearch 中的 Shard 数目直接反映 Spark 中的 RDD task 数目, 即任务量与分区数呈正相关关系, 如果采用默认分区方式会导致严重负载不均衡现象, 如很多分区块分布在负载高的数据节点上进行处理分析, 那么会拖慢整个作业完成效率, 因为 Spark 任务调度结束是所有 task 都完成的时刻. 现实应用中, 数据倾斜问题普遍存在, 由其引起的处理节点负载不均衡是 Spark-Elasticsearch 集成框架应用不可避免的问题.

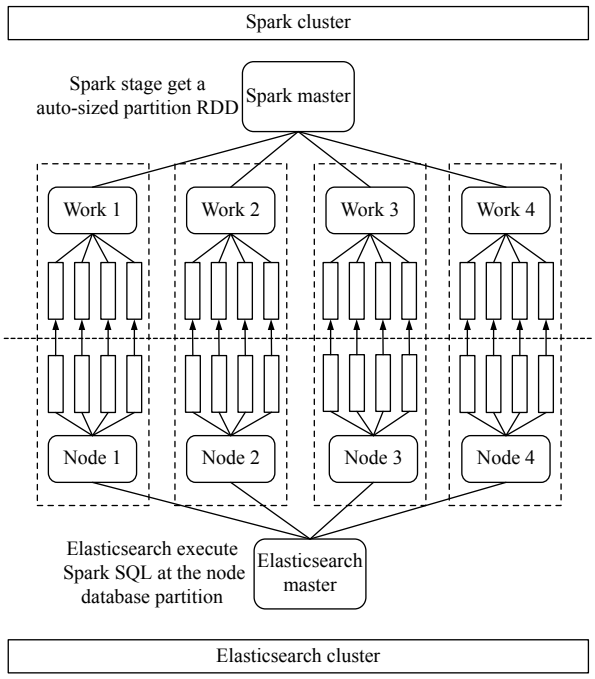

图 1 Spark 读 Elasticsearch 中数据机制图

有关动态分区解决负载均衡性的相关研究中, 李想等人[8] 提出一种改进的遗传算法的数据分配策略, 采用自适应交叉和变异算子策略. 王晓燕等人[9] 采用 Nash-Pareto 优化均衡策略协同自动数据分布. 王新友等人[7] 设计一种基于时间序列模型二次指数平滑法对引航事故进行预测, 减少引航事故的发生. 杨华芬[10] 提出一种基于优化 FA 模型的动态迁移算法, 首先构建动态迁移框架, 确定数据中心和网络节点的拓扑结构, 然后利用 FA 仿生群智能算法在数据中心区域范围内更新个体的位置并寻找最优解, 最后引入适应度函数和自适应惯性权重优化算法并扩大寻优范围, 实现大数据迁移成本最小化. 刘琨[11] 采用一次平滑预测算法+客观熵值法权重模型, 通过确定节点负载情况来进行虚拟机资源迁移达到负载均衡效果. 陈涛等人[12] 提出一种数据分布算法 CCHDP, 将聚类算法和一致性 Hash 方法结合, 根据设备可用资源权重自适应调节分配数据. 宋怀明等人[13] 设计了自适应散列和直方图相结合的数据分布策略, 动态调整节点负载均衡性. 王晶等人[14] 设计了一种基于动态阈值的迁移时机判决算法与基于负载类型感知的选择算法相结合的虚拟机动态迁移策略, 最终根据虚拟机与目的节点的资源匹配度与迁移代价选择目的节点, 对高负载与低负载节点的虚拟机动态调整, 从而优化节点资源配置问题.

针对 Spark-Elasticsearch 集成框架进行数据分析的应用场景, 需要提出 Elasticsearch 分区策略来改善负载均衡性, 提高应用的响应速度. 因此本文提出一种基于节点负载的数据动态分区机制和策略. 基于节点负载的数据动态分区机制包括负载监测采集、预测、数据预分区、数据迁移等模块; 基于节点负载的数据分区策略采用二次平滑法预测节点负载, 结合了 AHP 和熵值指标权重法, 能够根据不同的数据分析应用得到相应的分区策略, 动态地调整系统的负载均衡性, 提高应用的响应速度.

2 基于节点负载的数据动态分区机制和策略

针对 Spark-Elasticsearch 集成框架的应用场景, 即采用 local 方式读取数据分析的方式, 需要有效地将数据进行分布来提高系统的负载均衡性, 通过提高并行度来降低应用的响应时间. Spark 的 Worker 节点 local 方式并行地从 Elasticsearch 存储节点进行数据读写、计算分析, 如图 2 所示. Elasticsearch 中最小的物理单元是 Shard, 而 Elasticsearch 中的 Shards 数目直接反映 Spark 中的 RDD task 数目.

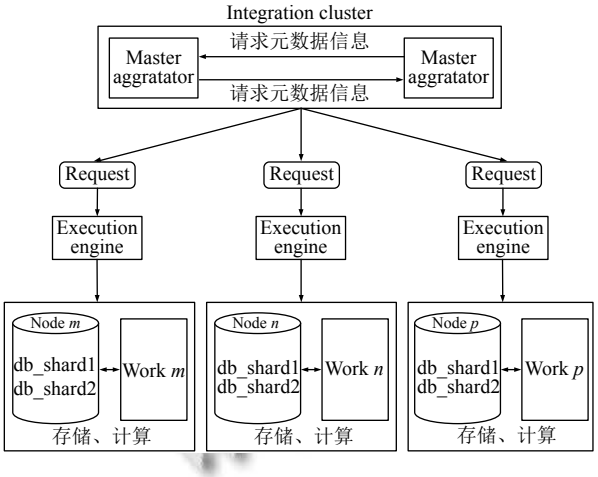

图 2 Spark-Elasticsearch 集成框架图

因为 Spark 并行访问 Elasticsearch 中的每个分区中的数据, 并将每个分区中的数据转化成自身的一个 RDD 分区, 由于 Spark 中的一个 RDD 分区对应一个 task, 而 task 需要 CPU 来处理, 为了减少 CPU 资源的上下文切换, 充分利用 CPU 资源, 则 Elasticsearch 中的 Shards 数目的设置要适中, 不能过小或过大, 经相关技术人员进行长时间的研究测试, 一般选择可用 CPU 核数的 2–3 倍; 同时每个 Shards 中的数据需近似相等, 来提高 task 的并行度.

在数据预分区阶段, Elasticsearch 的数据分布默认是每个节点的分区数目相同, 而这样会因为集群中每个节点资源的异构性导致节点处理能力的不同而引起节点之间的数据倾斜现象, 引发集群负载不均衡的问题. 集群节点偶然会出现负载瞬时高低峰值的情况而影响数据分区决策的拟定; 同时由于任务类型的不同, 如计算型、内存性、网络传输等任务类型会导致每种指标的权重不同, 需要确定每种指标权重来拟定数据分区策略; 如果数据已经分配完, 但集群在应用过程中可能会因为负载不均衡问题导致任务执行阻塞, 也可能因为节点的增删操作引发负载不均衡等问题, 在现实应用中都可能遇到.

因此, 为了解决上述问题, 需要设计相应的数据动态分区机制和策略来改善系统的负载均衡性来提高应用的响应速度.

2.1 数据动态分区机制

为了动态调整集群负载均衡性来提高应用的响应速度, 提出了基于节点负载的数据动态分区机制, 如图 3 所示, 该机制包括集群节点资源的监测、采集, 数据的预分区、迁移等模块. 整个 Spark-Elasticsearch 集成集群一直处于应用使用中, 监测模块定时读取从节点中各个指标的负载信息, 动态地显示 CPU、内存和带宽的利用率等信息; 然后通过资源采集模块将负载信息缓存起来, 并定期持久化到 MySQL 中, 为负载预测提供指标实时负载信息; 接着如果有大量数据进行导入时, 需要对每个节点的每种指标进行预测, 再通过指标权重判定方法获得每种指标的权重, 接着再依据负载预测后的指标信息与每种指标的权重值来获得每个节点的处理能力, 最后根据每个节点的处理能力进行数据分布, 完成数据的预分区; 如果集群在实际应用中出现严重的负载不均衡问题, 如达到了设定负载阈值, 则将高低负载节点加入到源、目标机队列中, 并结合迁移策略进行分区块的迁移.

2.2 数据动态分区策略

本文基于节点负载来进行数据有效地分布, 要解决如下问题:

(1) 初始情况下, 面对大量数据的导入, 数据预分区根据什么原则进行数据分布.

(2) 可能因为某些应用执行完或者预分区不合理情况导致集群负载不均衡现象, 通过数据迁移调整负载.

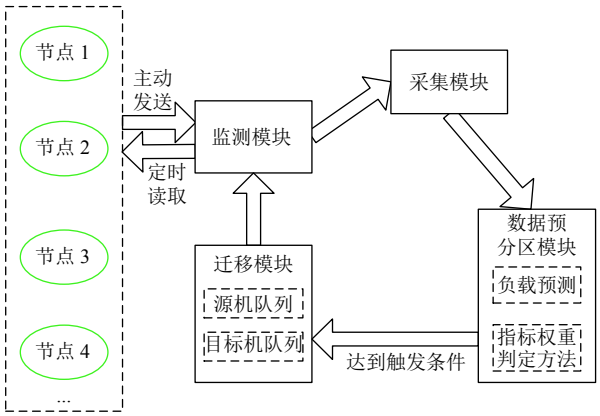

图 3 基于节点负载的数据动态分区机制流程图

本节通过动态分区策略来改善系统负载均衡性, 提高应用的响应速度. 采用二次平滑法预测节点负载, 结合了 AHP 和熵值指标权重法, 能够根据不同的数据分析应用得到相应的分区策略, 动态地调整系统的负载均衡性, 提高应用的响应速度.

2.2.1 数据预分区策略

(1) 负载预测

因数据预分区不合理、节点的宕机删除节点、增加节点进行水平扩展及出现负载极其不均衡问题等, 都需要进行分区块的迁移来平衡负载量, 采用二次指数平滑法负载预测模块来决定迁移量.

其中, n 代表取的周期数, j 代表第 j 个周期. 预测机制的流程如图 4 所示, 通过调整平滑系数 α 值来计算偏方差 S, 取 S 最小时对应的平滑系数 α 值. n、d 的值由用户设定.

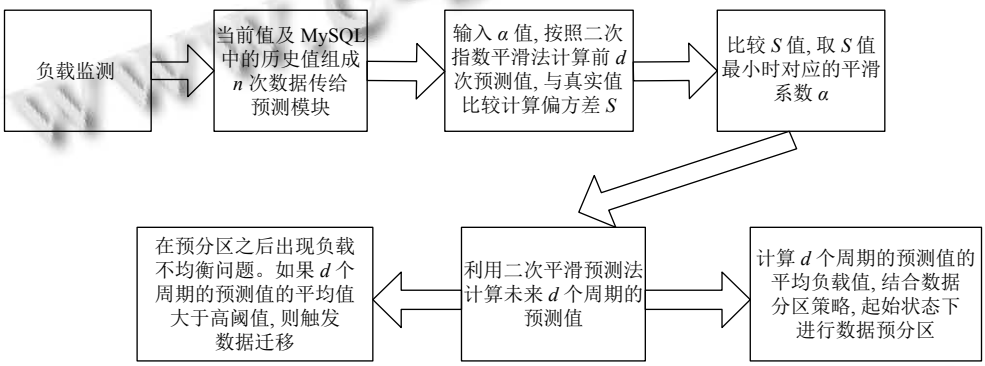

图 4 预测机制流程图

(2) 指标权重判定方法

因为 Spark-Elasticsearch 集成框架环境下的应用场景, 内存变化波动较小, 带宽和 CPU 变化波动较大, 如果只考虑客观熵值法则会影响内存的权重判定, 如果只考虑主观 AHP 权重法会忽略某些指标的重要性. 因此, 本文通过基于二次平滑负载预测法+主客观 AHP 与熵值指标权重集成法结合的指标权重判定方法算出每个节点的整体负载值, 最终再根据整体负载值来分配相应的数据分区数.

1) AHP

AHP 由决策者对所有评价指标进行两两比较, 得到判断矩阵 $U=(A_{ij})_{n\times n}$. 本论文取 CPU 利用率、内存利用率和带宽利用率作为负载的评价指标, 判断矩阵如下:

$$M=\begin{bmatrix} CUR_1 & MUR_1 & BUR_1 \\ CUR_2 & MUR_2 & BUR_2 \\ \cdots & \cdots & \cdots \\ CUR_n & MUR_n & BUR_n \end{bmatrix}$$

其中, A_1, A_2, A_3 分别代表 CPU 利用率、内存利用率和带宽利用率对节点整体负载影响的权重值. 对每列进行归一化求取特征向量, 再对每行进行归一操作求取特征向量, 最后得到各指标的权重配比, 同时对判断矩阵 A 进行一致性检验, 证明判断矩阵的合理性, 最终可得到 CPU、内存和带宽的主观权重分别为 WS_1, WS_2, WS_3, 并且 $WS_1+WS_2+WS_3=1$.

2) 熵值法

熵值法通过判断指标变化的离散度来反映该指标影响程度, 能够通过指标变异度客观地确定指标权重. 具体步骤如下:

① 通过测试过程得到各指标负载来构建负载信息决策矩阵 M

$$M=\begin{bmatrix} CUR_1 & MUR_1 & BUR_1 \\ CUR_2 & MUR_2 & BUR_2 \\ \cdots & \cdots & \cdots \\ CUR_n & MUR_n & BUR_n \end{bmatrix}$$

其中, n 代表周期数, CUR、MUR 和 BUR 分别代表 CPU、内存和带宽的利用率.

② 对决策矩阵 M 每列做归一化处理得到矩阵 R

$$R=\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ \cdots & \cdots & \cdots \\ R_{n1} & R_{n2} & R_{n3} \end{bmatrix}$$

其中, $R_{i1}=\dfrac{CUR_i}{\sum_{i=1}^{n}CUR_i}$, $R_{i2}=\dfrac{MUR_i}{\sum_{i=1}^{n}MUR_i}$, $R_{i3}=\dfrac{BUR_i}{\sum_{i=1}^{n}BUR_i}$,

矩阵 R 每一列满足归一性, 即 $\sum_{i=1}^{n}R_{ij}=1$, j=1,2,3, 即每一列值的和为 1.

③ 利用熵公式计算指标不确定度

用 E 表示某种指标的熵, 公式如下:

$$E_j=-K\sum_{j=1}^{n}R_{ij}\ln(R_{ij})$$

其中, E_j 代表指标的熵值, 常数 K=1/ln(n), 这样能保证 0≤E≤1, 即 E 最大为 1. 当某个属性下各值的贡献度趋于一致时, E 趋于 1, 可看出属性列值差异性大小可影响权系数大小, 因此可定义 D_j 为某个指标的贡献度, $D_j=1-E_j$.

④ 计算每种指标的客观权重值, 公式如下:

$$A=\begin{bmatrix} A & A_1 & A_2 & A_3 \\ A_1 & A_1/A_1 & A_1/A_2 & A_1/A_3 \\ A_2 & A_2/A_1 & A_2/A_2 & A_2/A_3 \\ A_3 & A_3/A_1 & A_3/A_2 & A_3/A_3 \end{bmatrix}$$

WO_1, WO_2, WO_3 代表内存、CPU 和带宽对于节点负载影响的客观权重值, 并且 $WO_1+WO_2+WO_3=1$. 先通过输入每种指标不同周期负载值矩阵, 再计算每种指标客观权重值, 最后通过熵值法计算得到每种指标的客观权重值.

3) 主客观 AHP 和熵值法权重集成法

真实情况可能出现主、客观指标权重设计的弊端问题, 为了减少弊端影响性. 因此本发明设计主客观集成的方法来解决此类问题, 平衡两者的权重偏差. 集成权重公式如下:

$$w_i=\beta\times WS_i+(1-\beta)\times WO_i$$

其中, β 为主客观权重调整系数, w_i 为最终节点负载的权重, 其中 i=1,2,3, 并且 $w_1+w_2+w_3=1$. 输入每种指标的主客观权重值, 再通过主客观权重集成法得到每种指标的集成权重值.

4) 节点的数据分布

首先, 由前面模块得到了内存、CPU、带宽 3 种指标在负载中所占的主客观集成权重大小后, 分别为 w_1, w_2, w_3.

然后, 通过每种指标权重计算得到每个节点的处理能力, 公式如下:

$$CA_i=w_1\times(1-CAU_i)+w_2\times(1-MAU_i)+w_3\times(1-BAU_i)$$

其中, CAU_i, MAU_i, BAU_i 分别代表指标预测后的 CPU、内存、带宽利用率, i 代表第 i 节点.

最后, 得出每个节点要分配的数据量的占比, 公式如下:

$$DP_i=\frac{CA_i}{\sum_{i=1}^{m}CA_i}$$

其中, DP_i 代表第 i 节点应分配的数据量占比, 有 m 个节点.

通过以上步骤后可知给集群中每个节点分配的数据量, 即相应的分区数.

2.2.2 数据迁移策略

通过设置高低负载阈值来作为触发数据迁移的条件, 构造出源机和目标机的选择队列. 在数据预分区之后出现负载不均衡问题或者增删节点的情况, 需要选择源机和目标机来进行数据迁移, 源机作为待迁移数据的节点, 目标机作为接受迁移数据的节点, 并获得应迁移的分区数.

(1) 源机选择

首先, 从负载缓存数组中读取内存利用率、CPU 利用率和带宽利用率负载信息采用二次平滑法进行预测, 预测 T 个周期后的每种指标平均负载值.

然后, 将每种指标的负载利用率预测值与主客观权重集成方法得到每种指标权重值相结合, 进而得到每个节点的整体负载值 $Load_i$. 负载值公式如下:

$$Load_i=w_1\times CUR_i+w_2\times MUR_i+w_3\times BUR_i$$

其中, CUR_i, MUR_i, BUR_i 和 w_1, w_2, w_3 分别为预测后的 CPU 利用率、内存利用率、带宽的利用率和权重值.

接着, 比较每个节点的负载值 $Load_i$ 和设置的阈值 H_{th}, 如果某个节点的负载值超过阈值 H_{th}, 则将该节点加入到高负载节点队列中.

然后, 按照 $Load_i$ 值由大到小构成源机选择队列

$S_{源}=\{s_1, s_2, \cdots, s_m\}$.

最后, 从 $S_{源}$队列中选取源机. 对 $S_{源}$队列中的 *Load* 值按降序排列, 按 $Load_i$ 值从大到小的顺序进行源机的选择.

(2) 目标机选择

首先, 从负载缓存数组中读取内存利用率、CPU 利用率和带宽利用率负载信息进行预测, 分别预测每种指标 T 个周期后的平均负载值.

然后, 将每种指标的负载利用率预测值与主客观权重集成方法得到每种指标的负载权重值结合, 代入主客观集成法计算, 进而得到每个节点的整体负载值 $Load_i$.

接着, 将每个节点的负载值 $Load_i$ 与设置的阈值 L_{th} 进行比较, 如果某个节点的负载值低于阈值 L_{th}, 则将该节点加入到低负载节点队列中.

然后, 按照 $Load_i$ 值构建由小到大目标机选择队列 $D_{目}=\{d_1, d_2, \cdots, d_m\}$.

最后, 从 $D_{目}$队列中选取目标机. 对 $D_{目}$队列中的 *Load* 值按升序进行排列, 按 $Load_i$ 从小到大的顺序进行目标机的选择.

(3) 迁移的分区数

1) 如果高低负载队列节点数相同, 即 $S_{源}=D_{目}$. 则分别将高低负载节点队列的数据按照顺序进行匹配并行迁移, 迁移的分区数公式如下:

$$N_{迁}=\left|\frac{N_{源}-N_{目}}{2}\right|$$

其中, $N_{迁}$, $N_{源}$, $N_{目}$分别代表迁移的分区数, 源机中的分区数, 目标机中的分区数.

2) 如果高负载队列节点数大于低负载节点数目, 即 $S_{源}>D_{目}$. 则适当调整低负载阈值, 使低负载节点队列节点数目等于或近大于高负载节点队列节点数目, 接着按照上述迁移公式设定迁移的分区数.

3) 如果高负载队列节点数目远小于低负载节点数目, 即 $S_{源}<D_{目}$. 则适当调整高负载阈值, 使高负载节点队列节点数目等于或近小于低负载节点队列节点数目, 接着按照上述迁移公式设定迁移的分区数.

4) 得到匹配的源机和目标机队列, 并知道了每组中源机应迁移的分区数, 采用并行进行迁移, 减少迁移开销.

通过以上步骤, 系统可实现负载均衡. 对于增删节点的突发情况, 同样可以采用此种迁移策略.

3 实验与分析

本文针对 Spark-Elasticsearch 集成框架的应用场景, 在局域网下部署 Spark-Elasticsearch 集成集群环境, 集群中每个节点都是虚拟机, 本部分实验 4 个节点, 相关配置参数如表 1 所示, 设定的总分区数为 32 个分区, 利用某制造企业中的数据集, 验证基于负载预测和 AHP、熵值集成权重法结合的数据动态分区策略的有效性.

表 1 Spark-MemSql 集成框架机器参数

节点名称	节点角色	内存 (GB)	CPU频率 (GHz)	节点带宽 (Mb/s)	系统
master	master, worker, data	8	3.2	300	Centos6.7
slave1	worker, data	6	2.7	200	Centos6.7
slave2	worker, data	5	2.5	250	Centos6.7
slave3	worker, data	4	2.2	300	Centos6.7

3.1 实验环境

3.2 实验数据

实验采用某制造业表 product 作为测试数据集, 如图 5 所示, 大约有 5000 万行数据. 每条数据主要包括产品长度、产品拉伸长度、产品重量等. 其中 weight 和 length 两列作为关联分析应用测试集, weight、drawlength、length 三列可作为 K-means 应用测试集, 不同的应用利用不同的数据集做测试.

TIMES	SUBTYPES	FACTO	LENGTH	DRAWLENGTH	WEIGHT	PRODNO
201611250559	FIBRE-Y-GI62.5-HEC-	老厂	5.1	5102.4	148	C3Y27426
201611201330	FIBRE-Y-MCSM-R47-	新厂	4.9	4934.3	136	R(A1M14912/A1M
201611071003	FIBRE-Y-MCSM-R47-	新厂	4.7	4743.4	128	R(A1N00089/A1N
201611240109	FIBRE-Y-MCSM-R47-	新厂	4.6	4623.2	132	R(A5M13966/A2N
201611160723	FIBRE-Y-MCSM-R47-	新厂	5	5102.4	143	R(A1A03740/A4F1
201611160932	FIBRE-Y-MCSM-R47-	新厂	5.4	5435.3	163	R(A6M14879/A5A
201611110819	FIBRE-Y-G657-RIC18	新厂	4.6	4639.7	128	R(SR(E3F16878P/E
201611271650	FIBRE-SHINETS-D18(	新厂	5	5012.8	142	KYOFSMM-01563
201611061702	FIBRE-Y-MCSM-R47-	新厂	4.4	4459.6	116	R(A5N00135/A2N
201611190313	FIBRE-Y-MCSM-R47-	新厂	4.9	4975.8	133	R(A2A03825/A6A(

图 5 分区策略性能对比测试数据集

3.3 基于节点负载的动态分区策略实验分析

(1) 对不同预分区策略进行性能对比测试. 本实验对 weight 和 length 两列做关联分析, 分析产品长度和重量之间的关联性; 对 weight、drawlength、length 三列做 K-means 聚类分析, 通过聚类分析分类. 通过比较默认预分区策略、负载预测+AHP 权重法、负载预测+AHP 与熵值集成权重法的 3 种不同预分区策略, 然后分别统计执行相同应用的时间, 验证方案的有效性.

(2) 在 Spark-Elasticsearch 框架中如果出现集群负载不均衡现象, 结合迁移策略对源机和目标机的数据

分区块进行迁移, 重复运行应用程序, 对迁移前后做性能对比, 验证方案的有效性.

3.3.1 预分区策略对比测试

通过不同的预分区策略进行分区, 实验重复运行应用程序, 第 1 组实验进行关联分析的应用; 第 2 组实验进行 K-means 聚类分析的应用. 对执行时间进行对比, 验证不同分区方案的有效性.

(1) 不同应用中不同指标平滑系数 α

采用二次平滑预测算法最终获得不同应用场景下的不同指标的平滑系数 α, 如表 2 和表 3 所示.

表 2 关联分析应用中不同指标的平滑系数 α

指标	CPU	内存	带宽
平滑系数α	0.7	0.35	0.75

表 3 K-means 聚类分析应用中不同指标的平滑系数 α

指标	CPU	内存	带宽
平滑系数α	0.75	0.40	0.65

(2) 利用 AHP 得出每种指标的权重

首先, 输入指标决策矩阵 A:

$$A=\begin{bmatrix} A & A_1 & A_2 & A_3 \\ A_1 & 1 & 2 & 9 \\ A_2 & \frac{1}{2} & 1 & 5 \\ A_3 & \frac{1}{9} & \frac{1}{5} & 1 \end{bmatrix}$$

评判采用列与行进行两两比较, 其中 A_1, A_2, A_3 分别代表 CPU、内存、带宽; 然后, 计算随机一致性比率 $C.R.=C.I./R.I.=0.001<0.1$, 说明决策矩阵设计合理; 接着, 利用 AHP 获得每种指标的权重值; 最后通过多次实验调整设置权重系数 β 为 0.7, 得到集成权重值, 不同的应用场景结果分别如表 4 和表 5 所示.

表 4 关联分析应用中不同负载指标权重值对应表 (%)

指标	CPU	内存	带宽
AHP权重值	61.523	31.872	6.605
AHP+熵值法权重值	57.762	29.62	13.518

表 5 K-means 聚类分析应用中不同负载指标权重值对应表 (%)

指标	CPU	内存	带宽
AHP权重值	61.523	31.872	6.605
AHP+熵值法权重值	58.762	30.52	11.518

(3) 根据应用中预测的每种指标负载值和不同权重方法, 获得不同分区策略下每个节点的处理能力, 得出每个节点分区数的分区数, 如表 6 所示.

通过图 6、图 7 所示, 分别执行关联分析和 K-means 聚类应用. 从整体上看默认分区策略效果最差, 本文设计的预测+AHP 与熵值权重集成法分区策略最好, 并且随着数据量的增加, 效果越显著. AHP 权重法是主观权重法, 没有根据实际应用场景进行权重配比, 有失客观性; Spark-Elasticsearch 框架的数据计算是在内存中进行的, 因此内存使用较稳定, 而带宽使用变化程度较大, 但利用率很低, 如果只采用客观法会导致内存权重小、带宽权重较大的错误结果. 因此集成主客观权重会带来更好的结果. 执行不同的应用取得了同样的效果, 说明本文研究的预分区策略在处理相对独立任务的应用上具有推广性.

表 6 不同分区策略推荐的每个节点的分区数

分区策略	Master	Slave1	Slave2	Slave3
默认分区方式	8	8	8	8
负载预测+AHP	12	10	5	5
预测+AHP和熵值权重集成法	11	10	6	5

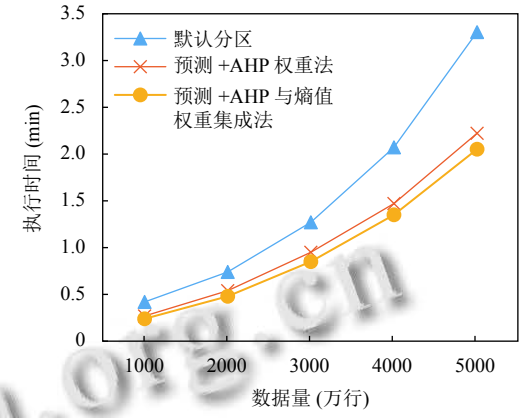

图 6 关联分析应用预分区策略性能对比图

如图 8、图 9 所示, 针对不同预分区策略执行相同的应用, 计算整个应用过程中每个节点平均负载利用率. 从整体上看默认分区策略出现严重的负载不均衡现象, 预测+AHP、预测+AHP+熵值权重集成法结合的预分区策略都能较好地解决集群负载问题, 实现集群负载的均衡性.

3.3.2 数据迁移对比测试

在 Spark-Elasticsearch 框架中遇到负载不均衡现象, 通过数据迁移策略, 对迁移前后的执行时间进行对比, 并考虑迁移时间开销, 验证方案有效性.

利用迁移策略构造高低负载队列, 获得不同节点应发送或接收的分区块数, 执行迁移操作后, 每个节点的分区数如表 7 所示.

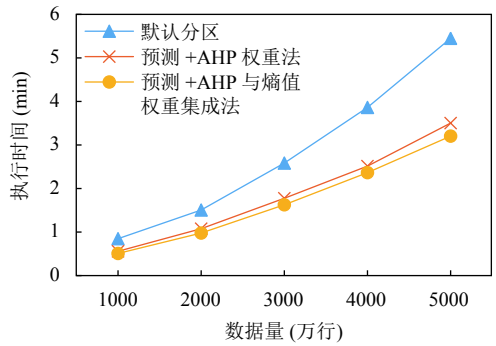

图 7　K-means 聚类分析应用预分区策略性能对比图

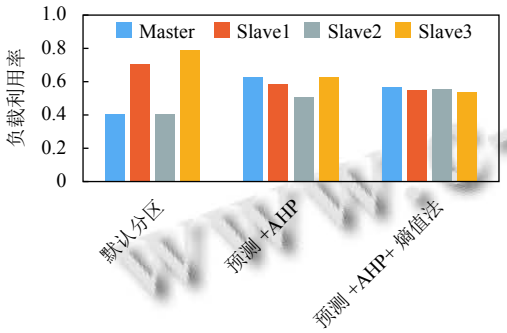

图 8　关联分析应用不同预分区策略节点负载利用率对比图

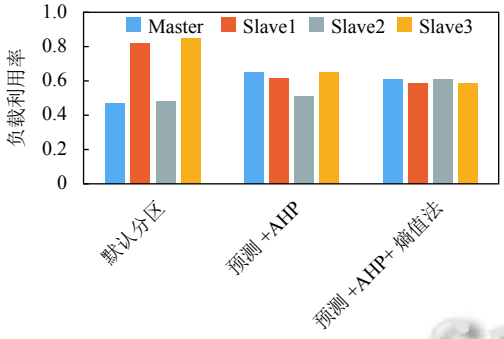

图 9　K-means 聚类分析应用的不同预分区策略节点负载利用率对比图

通过图 10、图 11 所示, 表明了迁移策略的有效性, 可以改善集群的负载均衡性, 一定程度上提高了应用的响应速度. 数据量较少时, 关联分析数据量小于 3000 万时和 K-means 分析数据量小于 2000 万时负载没到设定的阈值, 不触发迁移, 但是当数据量相对较大时, 即关联分析数据量达到 3000 万和 K-means 分析数据量达到 2000 万时负载超过阈值, 触发迁移. 虽然改善了集群的负载均衡, 但是迁移需要消耗时间, 导致总时间较长, 当数据量扩大时, 负载不均衡性加剧, 导致迁移成本相对较小, 提升了应用的响应速度.

表 7　迁移前后的节点分区数对应表

迁移前后	Master	Slave1	Slave2	Slave3
迁移前 (初始状态)	8	12	4	8
迁移后	11	10	6	5

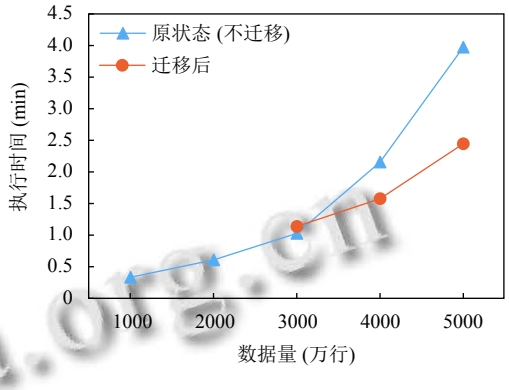

图 10　关联分析迁移策略性能对比图

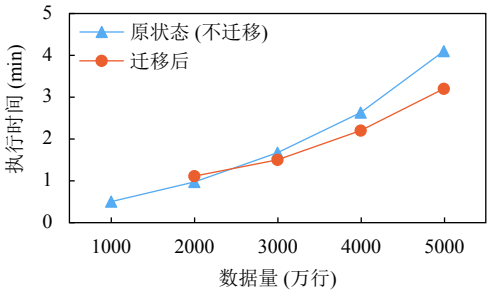

图 11　K-means 聚类分析迁移策略性能对比图

通过图 12、图 13 所示, 对不同应用进行迁移, 比较迁移前后整个应用过程中每个节点平均负载利用率, 结果展示通过迁移能改善集群负载均衡性.

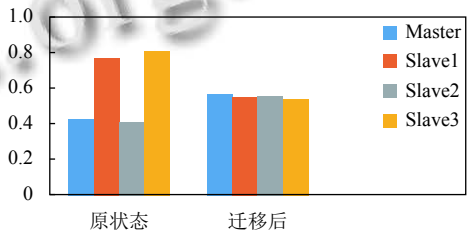

图 12　关联分析数据迁移前后节点负载利用率平均值对比图

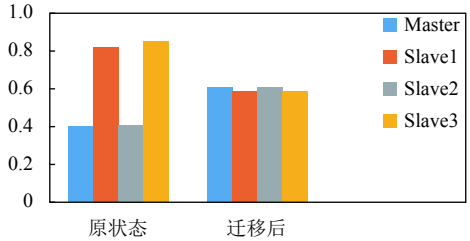

图 13　K-means 聚类分析数据迁移前后节点负载利用率平均值对比图

4 总结

本文虽然改善了集群负载均衡性, 提高了数据分析应用的响应速度, 但是依然存在一些不足之处. 日后的工作应该围绕以下几个方面进行研究:

(1) 指标权重的获得

针对不同的应用难以找到最佳指标权重值, 本文采用主客观权重集成法, 能较好地避免主客观引起的偏差, 但不能得到指标权重配比最优解, 指标权重需做进一步研究及大量实验验证.

(2) 数据迁移开销的理论研究

本文虽然通过数据迁移策略改善了集群负载均衡性, 但没从理论上考虑数据迁移过程中成本问题, 只是实验中包括了迁移时间.

参考文献

1 Vanhove T, van Seghbroeck G, Wauters T, *et al*. Managing the synchronization in the Lambda architecture for optimized big data analysis. IEICE Transactions on Communications, 2016, E99-B(2): 297–306. [doi: 10.1587/transcom.2015ITI0001]

2 He ZY, Huang QL, Li ZF, *et al*. Handling data skew for aggregation in Spark SQL using task stealing. International Journal of Parallel Programming, 2020, 48(6): 941–956. [doi: 10.1007/s10766-020-00657-z]

3 张占峰, 王文礼, 耿珊珊, 等. Spark 数据倾斜问题研究. 河北省科学院学报, 2020, 37(1): 1–7.

4 Dhulavvagol PM, Bhajantri VH, Totad SG. Performance analysis of distributed processing system using shard selection techniques on Elasticsearch. Procedia Computer Science, 2020, 167: 1626–1635. [doi: 10.1016/j.procs.2020.03.373]

5 黄秋兰, 程耀东, 陈刚. 分布式存储系统的哈希算法研究. 计算机工程与应用, 2014, 50(1): 1–4, 77. [doi: 10.3778/j.issn.1002-8331.1306-0307]

6 Li Y, Jiang YY, Gu J, *et al*. A cloud-based framework for large-scale log mining through Apache Spark and Elasticsearch. Applied Sciences, 2019, 9(6): 1114. [doi: 10.3390/app9061114]

7 王新友, 杨昆璘. 基于二次平滑指数的引航事故预测研究. 中国水运, 2017, 38(9): 56–57.

8 李想. 分布式数据库数据分配策略研究 [硕士学位论文]. 大连: 大连理工大学, 2009. 5–7.

9 王晓燕, 陈晋川, 郭小燕, 等. 基于 Nash-Pareto 策略的自动数据分布方法及支持工具. 计算机研究与发展, 2015, 52(9): 1965–1975. [doi: 10.7544/issn1000-1239.2015.20140832]

10 杨华芬. 云存储环境下大数据实时动态迁移算法研究. 机械设计与制造工程, 2021, 50(2): 117–122. [doi: 10.3969/j.issn.2095-509X.2021.02.025]

11 刘琨. 云计算负载均衡策略的研究 [博士学位论文]. 长春: 吉林大学, 2016. 85–110.

12 陈涛, 肖侬, 刘芳, 等. 基于聚类和一致 Hash 的数据布局算法. 软件学报, 2010, 21(12): 3175–3185.

13 宋怀明, 安明远, 王洋, 等. 大规模数据密集型系统中的去重查询优化. 计算机研究与发展, 2010, 47(4): 581–588.

14 王晶, 何利力. 基于虚拟机动态迁移的负载均衡策略. 计算机系统应用, 2020, 29(5): 167–174. [doi: 10.3969/j.issn.1003-3254.2020.05.024]