doi：10.3969/j.issn.2095-0411.2024.05.007

Spark 异构集群负载均衡调度策略

陶宇炜[1]，谢爱娟[2]

（1.常州大学 信息化建设与大数据处，江苏 常州 213164；2.常州大学 石油化工学院，江苏 常州 213164）

摘要：针对 Spark 可扩展分布式平台在作业任务调度时，没有考虑异构集群节点计算能力的差异和负载均衡问题，导致系统性能受到影响，文章构建了一种 Spark 环境下异构集群节点负载均衡调度策略。计算节点根据抽样算法，预测数据分布特征，将数据均衡划分为多个分区，根据异构集群节点静态负载和动态负载权重分配，获得异构集群节点实时负载，动态调度作业任务。最后，在异构集群上，通过 Wordcount，TeraSort，*K*-means 三种基准测试比较分析。实验结果表明，该算法运行时间明显减少，异构集群的性能得到提升。

关键词：异构性；作业调度；负载均衡；Spark

中图分类号：TP 302 **文献标志码**：A **文章编号**：2095-0411(2024)05-0061-10

Load balancing scheduling policies for Spark heterogeneous clusters

TAO Yuwei[1]，XIE Aijuan[2]

（1. Office of IT Services and Big Data，Changzhou University，Changzhou 213164，China；2. School of Petrochemical Engineering，Changzhou University，Changzhou 213164，China）

Abstract：Aiming at the problem that the Spark scalable distributed platform does not consider the computing capabilities of heterogeneous cluster nodes and load balance during job task scheduling，which affects the system performance，this paper constructs heterogeneous cluster nodes load balance scheduling policy under the Spark environment. Heterogeneous cluster node predicts the data distribution characteristics according to the sampling algorithm，divides the data into balancing partitions. According to the static load and dynamic load weight distribution，heterogeneous cluster node obtains the real-time load，and dynamically schedules job tasks. Finally，Wordcount，TeraSort，and *K*-means three benchmark tests were used to compare and analyze during heterogeneous cluster operation. Experimental results show that this algorithm can reduce the execution time significantly，and

收稿日期：2024-02-20。

基金项目：2021 年江苏省教育科学"十四五"规划立项课题资助项目(D/2021/01/131)；2021 年常州大学石油化工学院教育教学研究课题资助项目(SHJY202101)。

作者简介：陶宇炜(1968—)，男，江苏常州人，硕士，高级实验师。E-mail：tyw@cczu.edu.cn

引用本文：陶宇炜，谢爱娟. Spark 异构集群负载均衡调度策略[J]. 常州大学学报(自然科学版)，2024，36(5)：61-70.

improve the performance of heterogeneous cluster.

Key words: heterogeneous; job scheduling; load balancing; Spark

计算集群的异构主要是节点的 CPU、内存和 I/O 等部件的性能不同导致计算能力出现差异，节点负载不均衡和部分资源闲置状况影响了集群整体运行效率。面对多样化海量数据采集、存储和分析应用需求，Hadoop[1]，Storm[2]和 Spark[3]等分布式数据并行存储和处理框架符合数据处理的性能和效率需要。Hadoop 或 Spark 的任务调度策略，都是基于同构集群的理想化模块分配作业任务[4]。未考虑异构集群环境下作业任务的异构性问题及节点资源使用和负载变化，在异构集群环境下作业调度策略暴露出较严重的短板效应，从而影响异构集群运行效率。

学者们就提升 Hadoop 和 Spark 异构集群计算节点的全部性能开展了广泛研究。郑晓薇等[5]针对集群节点的计算能力提出 Hadoop 集群自适应任务调度，依据计算节点的负载状况，将任务特征和节点性能等作为任务调度依据，自动调整任务的分配。XU 等[6]根据 Hadoop 异构集群计算节点负载动态变化情况和作业任务的节点性能上的差异，提出基于动态调整工作负载的自适应作业任务调度策略，优化异构集群的性能。为了尽可能减少计算节点的资源争用，YONG 等[7]提出节点资源感知的任务调度策略，通过主节点收集并分析工作节点资源消耗情况进行作业任务调度。针对异构集群不能适用 Spark 原有的调度策略问题，徐佳俊等[8]提出分层调度策略，该调度策略对集群系统整体性能提升虽有明显效果，但没有考虑应用程序类型和节点处理能力的偏向性。胡亚红等[9]针对 Spark 默认任务调度算法没有考虑集群异构导致的资源不均衡问题，提出节点优先级动态自适应任务调度算法，动态调整作业任务运行过程中的节点优先级，完成作业任务的分配，缩短作业运行时间。

针对异构集群各节点的计算能力差异，异构集群作业任务调度过程中分配给工作节点的任务负载不均衡导致部分工作节点处于空闲状态及资源利用率低的问题，作业任务分配时存在资源浪费与内存溢出导致作业任务运行时间过长的问题，提出异构集群作业任务均衡调度策略。将异构集群按计算节点性能分区，利用集群各节点负载变化状况为作业任务配置资源，根据分区负载调度作业任务。对分布式计算框架 Spark 的内存分配机制和任务调度策略进行优化，设计均衡划分作业任务并分配作业任务的方法，把作业任务调度到与其相适应的工作节点，实现执行相应作业任务节点的负载均衡，减少作业任务运行时间，提高集群资源的利用效率。

1 Spark 架构及运行机制

Spark 框架的主从架构如图 1 所示。在 Spark 分布式计算框架自带的 Standalone 运行模式下[10]，Cluster manager 称为主节点（Master node）。主节点（Master node）接收用户端（Client）的作业提交，协调、管理工作节点（Worker node），为工作节点（Worker node）分配 CPU、内存等资源，启动 Driver program 和 Executor，并负责 Apache Spark 正常运行、资源和应用管理。工作节点（Worker node）用于执行作业任务，常驻 Worker node 守护进程，管理工作节点的计算资源，启动作业任务的执行者 Executor，负责节点中 Executor 和 Master 节点之间的通信。

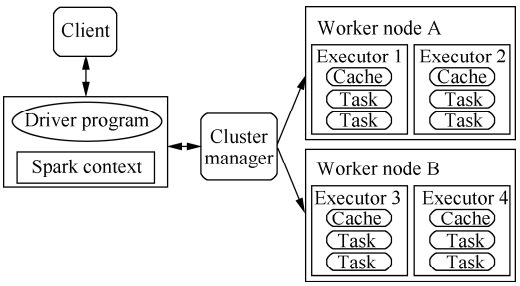

图 1 Spark 框架结构

Fig.1 Spark framework structure

2　Spark 异构集群负载均衡调度设计

计算集群的异构化现象导致集群计算资源使用的不均衡，不同类型的作业任务，如计算密集型作业任务和 IO 密集型作业任务对于集群计算资源的需求表现出明显的差异性，异构集群节点接收并行作业的任务量和作业任务的复杂度致使作业任务的运行也会相互争用集群中的计算资源。对于计算集群系统，数据对应作业任务，数据的不均衡分布即数据倾斜[11]。

借鉴 MapReduce 数据处理机制，在 Spark 框架中的 MapReduce 过程通过 Shuffle（混洗）连接 Map 和 Reduce 阶段。MapReduce 过程中的所有中间数据表示为键和值＜Key，Value＞的集合。Shuffle 是对数据进行重新分组。Spark 框架按照计算集群中存储的数据特征重新分组，将不同分组的数据聚合归入相同的位置计算，这个重新分组的过程称之为 Shuffle[12]。Shuffle 过程在 Spark 框架中是不可或缺的一个执行过程，Spark Shuffle 由 Shuffle Write 和 Shuffle Read 组成：在 Shuffle Write 中，数据由 Map Task 形成中间数据，再按数据分区方式填充到相应的分区（Partition），同一分区数据为具有相同 Key 的数据；在 Shuffle Read 中，Read Task 读取上游的分区数据。在进行 Shuffle 时，工作节点具有相同 Key 的数据聚合形成规模很大的数据簇（Cluster）分配给 Reduce 任务处理，就出现了数据倾斜现象，致使计算集群的计算效率大为降低。

为了解决 Spark Shuffle 过程的数据倾斜问题，从以下方面考虑：① 对集群工作节点（Worker node）上以键值对＜Key，Value＞形式出现的中间数据采样；② 预估通过采样的所有中间数据分布状况，并估算数据簇（Cluster）大小；③ 根据预估的中间数据分布，制定中间数据动态分区策略；④ 按照中间数据动态分区策略实施数据分区，执行作业任务分配，如图 2 所示。

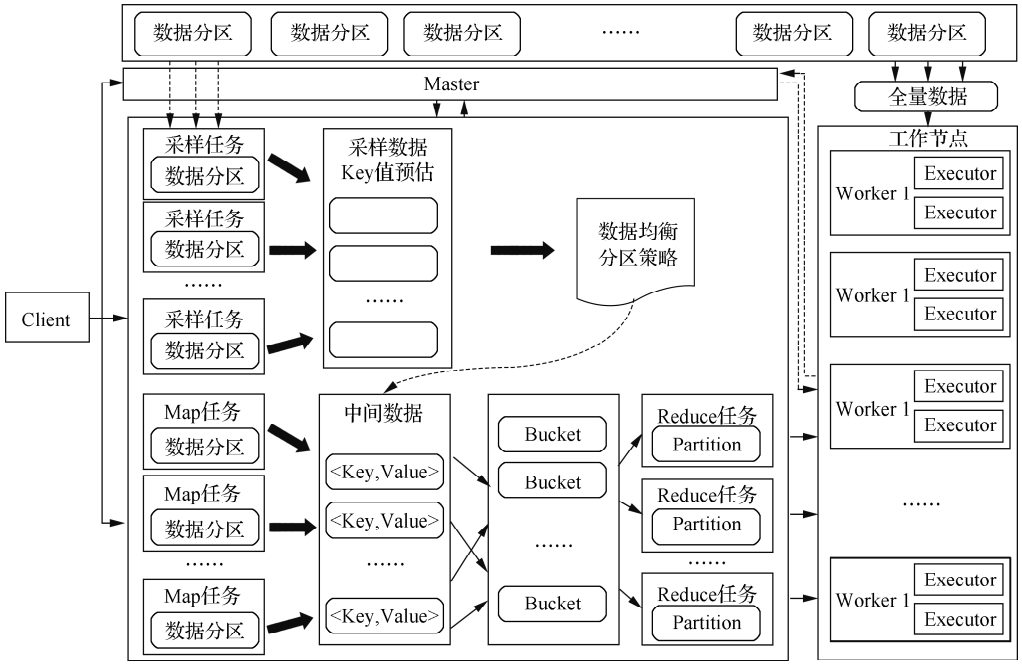

图 2　总体架构

Fig.2　Overall architecture

图 2 中总体架构包含：数据采样预处理、中间数据分布预估、数据均衡分区策略生成和应用。

1）数据采样预处理。为了避免 Reduce Stage 产生数据分区倾斜，在 Shuffle 之前预估中间数据键值对＜Key，Value＞的分布，对分布在工作节点（Worker node）的中间数据采用整群抽样算法执行采样任务。

2）通过采样得到样本数据的基础上，将各个工作节点（Worker node）采样信息以消息通信机制传送到主节点（Master node），预估中间数据的键频率分布。首先将采样后得到的各个分区数据进行汇总，预估弹性分布式数据集（RDD）的整体键分布，再依据 Map 端聚合发生与否，预估中间数据的键频率分布，估算数据簇（Cluster）的大小。

3）数据均衡分区策略生成。根据预估的中间数据键频率分布，执行中间数据均衡分区策略。根据中间数据出现的重度和轻度倾斜情况，采用均衡分区算法，使 Reduce 阶段任务分配的中间数据均衡，各工作节点（Worker node）负载均衡化，提高集群的运行效率。

4）数据均衡分区策略应用。在 Shuffle Write 数据阶段，按照数据均衡分区策略，对＜Key，Value＞键值对进行 Map 任务分区，计算结果按 ID 号依次写入磁盘，同时产生索引文件和数据文件，同一 Reduce 索引号的区域归入同一个数据段（Segment）。在 Shuffle Read 阶段，Reduce 任务按照索引文件将相同 Reduce 索引号的键值对＜Key，Value＞组成的 Reduce 分区，从 Map 任务执行的工作节点（Worker node）拉取数据进行 Reduce 任务处理。

2.1 数据采样

为确定中间数据分布状况，需要对全量数据进行采样。合适的采样算法可以对全量数据的总体分布情况进行有效评估，采样算法的选取需要结合不同的计算场景择优选择。从总体中采集样本的方式有两种，一种是概率抽样，即按照概率论原理，等概率随机抽取各个样本，这种方式可以有效避免抽样中人为影响因素，保证抽取样本的客观性；同时通过计算控制样本数据的抽样误差，准确说明样本数据的统计结果对于整体数据的适合度，进而推断整体数据的性质与分布情况。另一种是非概率抽样，样本数据的抽取由调查人员主观经验决定，不能严格遵守随机抽样原则，无法评估样本数据的抽样误差和抽样算法的准确率。现实中输入数据分布于各个工作节点，总体数量未知，而且具有随机性，因此，选取概率抽样算法中的抽样方式作为中间数据的评估方法。文章选择整群抽样算法，理由为：整群抽样将整体数据划分为集群中的一个个小群体，集群内部具有异质性特征，而小群体具有同质性特点，集群中的小群体有同等机会成为样本数据的一部分。Spark 中的 RDD 各分区类似于集群中的若干小群体，对 RDD 各分区的抽样方式就是整群抽样。再者，从若干小群体随机选择和抽样类似于模仿随机抽样，表明抽样结果的有效性。因此，选择整群抽样按预先设置比例抽取样本，对整体数据进行评估。

抽样算法过程描述为：

1）分别计算各个工作节点的 RDD 数据分区中各分区记录数，传送给驱动程序（Driver programmer），得到总记录数。

2）按照预先设置的抽样率计算抽样所需要的样本总数，并根据 RDD 分区数计算待抽样 RDD 中各分区样本数。抽样率设置越大，数据分布估算也就越准确，抽样任务耗时也越长。

3）各工作节点按照样本数对 RDD 分区执行抽样，统计样本 Key 值数量，采样结果以消息通信机制传送主节点（Master node）。

4）主节点（Master node）根据各个计算节点的本地抽样信息，结合预先设置的抽样率，统计总体数据的分布，即确定中间数据的分布特征。

抽样算法分别在每个计算节点针对 RDD 的各个分区执行。各节点的采样样本数据聚合结果发送给主节点（Master node），主节点（Master node）估算 RDD 分区键值对（K_i，C_i）数量。采样所涉网络传输量较少，采样过程速度较快，抽样算法步骤如下。

Input：键值对数据 RDD，RDD 分区数 rddPartitions，采样率 α

Output：tuples={(K_1，C_1)，(K_2，C_2)，…，(K_m，C_m)}

Step 1：计算总体数据大小 rddSize；

Step 2：sampleSize=rddSize ×α //计算总体数据的采样大小；

Step 3：sampleSizePartition = math. ceil（sampleSize/rddPartitions）//计算 RDD 分区的样本大小；

Step 4：调用 Sample 函数对各个 RDD 分区抽样，得到各个 RDD 分区样本；

Step 5：对本地节点（Local node）样本统计数据聚合，通过消息通信机制传送给主节点（Master node)；

Step 6：主节点（Master node）汇总各个计算节点的抽样样本数据；

Step 7：根据采样率确定数据样本分布情况 tuples。

2.2　异构集群区域划分

任务在异构集群各个计算节点上运行处理速度各不相同，各计算节点上的 Spark 作业运行时间不同。将运行时间相同或者相近的计算节点归并，将集群中的计算节点按异构性划分为不同的组，集群的异构性体现为不同性能的计算节点执行相同 Spark 作业的运行时间上的差异，将不同性能的计算节点划分成不同区域。

除此之外，还需要衡量不同区域计算节点的单个 CPU 核心能力（C_{cpu}）。在保证分配给各计算节点的单个 CPU 核心的作业任务类型和数量相同的前提下，各个区域参与测试的计算节点运行的 BR（Benchmark result，B_{r}）的任务并发度 TC（Task concurrency，T_{con}）相同，即为各工作节点（Worker node）CPU 核数 N_{cpu}的最小公倍数 LCM（Least common multiple，L_{cm}），见式（1）。从而保证每个区域计算节点的单个 CPU 核运行的作业数相同，得出各个区域计算节点运行时间，这样就较为准确地反映出不同区域计算节点的单个 CPU 核的性能差异，见式（2）。

$$T_{\text{con}} = L_{\text{cm}}\left(\sum_{\text{i=1}}^{\text{n}} N_{\text{cpu},i}\right) \tag{1}$$

$$B_{\text{r}} = \frac{\text{Spark}\left(N_i \times \dfrac{T_{\text{con}}}{N_{\text{cpu},i}},\ j_{\text{bm}}\right)}{T_{\text{con}}} \tag{2}$$

式中 j_{bm}为运行 Benchmark 基准程序的作业。

综上，集群异构性体现在 CPU 核的数量和 CPU 的运行能力差异，定义集群异构性为不同计算节点的 CPU 核数量 N_{cpu}、不同计算节点的计算能力 C_{nod}和单个 CPU 核心能力 C_{cpu}，见式（3）。

$$\text{Heterogeneity}\ (N_{\text{cpu}},\ C_{\text{nod}},\ C_{\text{cpu}}) \tag{3}$$

区域定义为集群中异构性相同的计算节点 N_i 分成的组，见式（4）。

$$\text{Zone}(\text{Heterogeneity}, \sum_{i=1}^{n} N_i) \tag{4}$$

式（3）中，C_{nod}建立基准程序 Benchmark，Benchmark 作业选取典型的 WordCount 程序对文本文件中的文本数据统计词频，Benchmark 作业运行结果 Benchmark result 为程序运行时间，程序运行时间与运行速度呈现反比关系，$1/B_{\text{r}}$ 反映节点的计算能力。节点的 BR（B_{r}）计算公式为节点运行 Spark benchmark 作业，见式（5）。

$$B_{\text{r}} = \text{Spark}\ (\text{Node},\ j_{\text{bm}}) \tag{5}$$

引入基准程序 Benchmark 后，同一区域节点的 CPU 数和 Benchmark result 都相同，则同一区域节点的 CPU 具有相同性能。

2.3 数据均衡分区

通过整群抽样算法产生的中间数据键值对集合为 tuples＝｛（K_1，C_1），（K_2，C_2），…，（K_i，C_i）｝，然后进行数据分区。利用集群异构性设计计算资源分配策略，根据用户对资源的使用需求，分区域进行作业调度，分配计算能力相同或相似的节点运行作业，合理使用集群资源，缩短作业运行时间，提高作业运行速度。具体的数据分区策略描述为：

1）按照中间数据键值对集合中的 C_i 值降序排列 tuples 中的数据键值对（K_i，C_i）；依次取出 Key，将最大键值对（K_i，C_i）中的 K_i 对应元素分配给拥有最大计算能力的节点所在的分区 Partition（P_j），如果目标分区域满足作业调度所需的资源分配需求，资源分配完毕。

2）如果目标分区域计算资源未满足作业调度资源分配要求，选择近目标分区域的较低等级计算能力的节点所在区域分配资源，过滤级别高的区域，即 filter（Zone. lever＜Spark. Zone. lever＞），以减小作业调度获取的资源异构性差异。低等级区域按照单个 CPU 核计算能力高低排序，单个 CPU 核计算能力相同区域则按照单个 CPU 核数量多少排序，保存排序结果在队列 Zone _ sortedquene 中，之后分配资源给作业，以此循环过程至计算资源分配结束。

3）如果较低等级计算能力的节点所在区域计算资源不够分配，则拆分键值对（K_i，C_i）。标记为分区标志 K_{i-1}，依此拆分 K_i 剩余数据分配给对应分区，至分配结束。在拆分键值对（K_i，C_i）过程中，标记对应分区标志 K_{i-2}，K_{i-3}，…，K_{i-m}，$1\leqslant m\leqslant n$，n 为分区数。

4）键值对（K_i，C_i）进行 K_i 拆分之后，相应的 C_i 同样转化为 C_{i-1}，C_{i-2}，…，C_{i-p}，$1\leqslant p\leqslant n$，以使数据一致，形成（K_i，C_i）转换为（K_{i-1}，C_{i-1}），（K_{i-2}，C_{i-2}），…，（K_{i-m}，C_{i-p}），$1\leqslant m\leqslant n$，$1\leqslant p\leqslant n$，n 为分区数。如果 Key 值存在于抽样集合，根据前述记录的 K_i 或 K_{i-1} 分区归属将其分配给相应分区，否则按照哈希算法分配键值对（K_i，C_i）。

完成作业计算任务后，如果存在 $K_i\rightarrow K_{i-1}$，K_{i-2}，…，K_{i-p} 转换关系，最终需依据记录 K_i 的分区归属标志合并计算结果。

至此，完成了均衡划分各个分区数据。但是，若出现分区数量少于作业运行所预先分配的 CPU 核的数量，就会形成 CPU 核处于空闲状态，导致作业运行时间延长。为此，通过调整分区数量，使分区数量为作业运行所预先分配的 CPU 核数的倍数，可以最大化利用集群计算资源。测试表明，产生的额外开销微不足道。

2.4 Spark 异构集群节点负载

在异构集群环境下，通过计算衡量集群节点负载大小，为节点分配相应资源，确保提交的作业按时完成，提高集群整体的运行效率。衡量集群节点负载情况，不仅与节点静态负载相关联，而且与作业运行时的动态负载相关联。

1）集群节点的静态负载

Spark 异构集群节点以 CPU 和内存为主要计算资源执行作业，得到集群节点 i 的平均运行时间 T_i，并进行归一化处理，以准确反映节点 i 静态负载大小 S_i。

$$S_i=\frac{T_i-\min(T_i)}{\max(T_i)-\min(T_i)}\quad i=1，2，3，\cdots，n \tag{6}$$

式中 T_i＝｛T_1，T_2，…，T_n｝。

2）集群节点的动态负载

Spark 异构集群中的应用程序主要利用 CPU 和内存两种计算资源执行作业任务，集群节点的负

载随着应用程序的运行状态而产生动态变化。为了实时反映集群计算资源的使用状况，采用 CPU 利用率 $U_{\mathrm{cpu},i}$、内存利用率 $U_{\mathrm{mem},i}$ 和作业队列长度量化 Spark 异构集群节点动态负载。为使作业队列长度统一成百分制形式，将作业队列长度采用任务竞争程度表示。

Linux 系统的内存文件系统以 proc 为内核和进程提供通信接口，动态地从系统内核采集节点数据信息。proc 以内存文件方式提供的节点数据随着时间动态变化，以反映 Linux 系统中的各进程的状态变化。Linux 系统用 proc/stat 命令获取 CPU 用户态执行时间 $T_{\mathrm{use},i}$、系统内核执行时间 $T_{\mathrm{sys},i}$ 和空闲系统进程执行时间 $T_{\mathrm{id},i}$ 和作为节点 CPU 执行的总时间 $T_{\mathrm{tot},i}$。再取相邻近两个时间间隔连续采样空闲时间 $T_{\mathrm{id1},i}$，$T_{\mathrm{id2},i}$ 和 CPU 执行的总时间 $T_{\mathrm{to1},i}$，$T_{\mathrm{to2},i}$ 计算得到 CPU 利用率 $U_{\mathrm{cpu},i}$。

$$T_{\mathrm{tot},i}=T_{\mathrm{use},i}+T_{\mathrm{sys},i}+T_{\mathrm{id},i} \tag{7}$$

$$U_{\mathrm{cpu},i}=1-\frac{T_{\mathrm{id2},i}-T_{\mathrm{id1},i}}{T_{\mathrm{to2},i}-T_{\mathrm{to1},i}} \tag{8}$$

调用 Linux 系统的 proc/meminfo 命令获取异构集群可用内存数 $T_{\mathrm{mT},i}$ 和剩余内存数 $T_{\mathrm{mR},i}$，得到异构集群节点 i 内存利用率 $U_{\mathrm{mem},i}$

$$U_{\mathrm{mem},i}=1-\frac{T_{\mathrm{mR},i}}{T_{\mathrm{mT},i}} \tag{9}$$

工作节点（Worker node）执行作业任务，Linux 系统用 proc/loadavg 命令获取节点在 1 min 作业队列长度的平均值 $L_{\mathrm{tas},i}$，与节点 i 的 CPU 核数 $N_{\mathrm{cor},i}$ 相除得到集群节点 i 任务竞争程度 $C_{\mathrm{tas},i}$

$$C_{\mathrm{tas},i}=\frac{L_{\mathrm{tas},i}}{N_{\mathrm{cor},i}} \tag{10}$$

上述 CPU 利用率 $U_{\mathrm{cpu},i}$、内存利用率 $U_{\mathrm{mem},i}$ 和节点任务竞争程度 $C_{\mathrm{tas},i}$ 三者的累加和，表示集群节点的动态负载。节点 i 的动态负载 D_i 表示为

$$D_i=U_{\mathrm{cpu},i}+U_{\mathrm{mem},i}+C_{\mathrm{tas},i} \tag{11}$$

综合衡量集群节点负载情况，异构集群节点 i 的负载表示为 $L_i=\alpha S_i+\beta D_i$，其中 α 和 β 分别为节点 i 的静态负载的权重值和动态负载的权重值，α 和 β 之和为 1。

2.5　异构集群节点负载均衡任务调度

为了提高异构集群性能，设计异构集群节点负载均衡任务调度策略。以包含两个工作节点的集群为例，整体架构如图 3 所示。

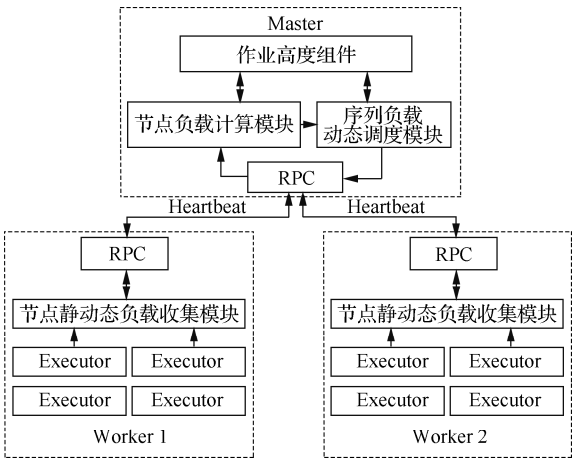

图 3　Spark 负载均衡调度策略设计架构

Fig.3　Design architecture of Spark load balance scheduling policy

工作节点 Worker 中的节点静动态负载收集模块主要负责检测节点负载和节点资源使用状况，将采集的负载指标数据值通过 RPC（Remote procedure call）传递给集群主节点 Master；集群主节点 Master 中的节点负载计算模块通过读取配置文件获得集群节点静态负载大小，利用工作节点 Worker 传递的数据计算动态负载值，再利用负载计算公式计算节点的负载大小。集群主节点 Master 中的序列负载动态调度模块负责初始化形成按负载大小排序的序列负载队列，并响应作业任务调度请求。Master 节点按照负载序列前后顺序分发作业任务给工作节点 Worker。集群中作业运行结束后，主节点 Master 利用 RPC（Remote procedure call）并结合心跳（Heartbeat）信息机制获取工作节点 Worker 的资源状况，通过重

新计算节点负载大小，更新序列负载队列，为处于等待队列中的作业任务分配资源量。

异构集群节点负载均衡任务调度算法具体如下。

输入：等待调度的作业集 TaskSet，任务数为 k，异构集群节点负载队列 worker _ Loader _ queue

输出：作业 Task 调度给执行器 Executor

Step 1：update worker load in worker _ Loader _ queue //更新异构集群工作节点负载队列

Step 2：for worker in worker _ Loader _ queue do

Step 3：worker. load/＝zone. cpu _ capacity //按单个 CPU 核计算能力处理异构集群工作节点负载

Step 4：end for

Step 5：resort worker _ Loader _ queue //异构集群节点负载队列重新排序

Step 6：sorted _ executor _ queue ＝ sort By Worker（worker _ Loader _ queue）//获得 Executor 队列

Step 7：for LOCALITY in data _ locality do //本地数据优先

Step 8： for executor in sorted _ executor _ queue do

Step 9： while executor can allocate task do //判断 Executor 能否启动作业

Step 10： task＝task At Locality（taskset，LOCALITY，executor）//确定等待调度作业

Step 11： Taskset. remove（task）//移除作业集中已调度作业

Step 12： allocate and start task at executor //分配 Executor 运行作业

Step 13： executor. cores－－＝spark. task. cpus //移除已分配的计算资源

Step 14： if executor. cores＜spark. task. cpus then //当前 Executor 是否有 CPU 启动作业

Step 15： resort worker _ Loader _ queue //负载队列重新排序

Step 16： sorted _ executor _ queue＝sort By Worker（worker _ Loader _ queue）//获取执行队列

Step 17： end if

Step 18： end while

Step 19： end for

Step 20：end for

上述为代码描述异构集群节点负载均衡任务调度。由于分布式调度涉及多个节点执行多个作业任务，采用多个循环嵌套，通过执行多次循环分配作业任务，发挥节点的计算能力，实现较为合理的作业任务调度。

3 实验与结果分析

异构集群工作节点的静态负载大小通过实验得知，进而确定工作节点静态负载和动态负载二者的权重大小，α 为 0.4，β 为 0.6。本节通过实验进行测评和比较，验证 Spark 异构集群并行调度算法的有效性。

3.1 实验环境搭建

通过安装 VMware 虚拟机构建 Spark 异构集群，每台 VMware 虚拟机作为异构集群的某个节点。部署的 Spark 异构集群包括主节点 Master、工作节点 Worker 1、Worker 2。集群参数配置见表 1。

表 1 集群参数配置

Table 1 Cluster parameter configuration

项目名称	参数配置
操作系统	CentOS 7
Spark	Spark2.1.0
主节点 Master	双核，4 GB 内存，50 GB 硬盘
工作节点 Worker 1	
工作节点 Worker 2	

实验利用 BigDataBench 的工作负载 Wordcount，TeraSort 和 *K*-means[13]，测试集群 Worker 节点性能。这些工作负载简单且易于理解，能够代表基本的 Spark 应用程序，应用较为广泛。

3.2　同种作业任务不同规模数据量实验

实验选取 Wordcount 和 TeraSort 处理作业任务，分别使用 Spark 默认调度算法和本文算法检测 Spark 异构集群运行效率是否有所提升。作业任务采用 2，4，6，8，10 GB 不同规模数据集，每个规模数据实验 10 次，并计算平均值，实验结果如图 4 和图 5 所示。图 4 给出了 Wordcount 负载实验结果，图 5 给出了 TeraSort 负载实验结果。

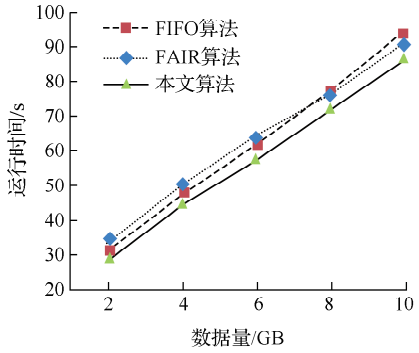

图 4　Wordcount 负载不同数据量运行时间对比

Fig.4　Comparison of the running time of Wordcount load with different amount of data

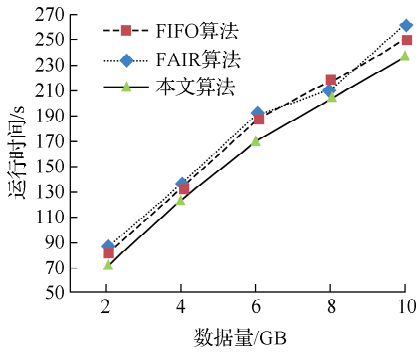

图 5　TeraSort 负载不同数据量运行时间对比

Fig.5　Comparison of the running time of TeraSort load with different amount of data

由图 4 可知，运行 Wordcount 工作负载时，使用本文算法相比较其他两种算法，任务的运行时间缩减较为显著。在 5 种不同数据量下，本文算法相较于 FIFO 算法，Wordcount 作业时间分别缩减了 7.77%，7.62%，7.22%，6.98%和 9.08%，平均缩减了 7.73%；本文算法相较于 FAIR 算法，Wordcount 作业时间缩减了 17.22%，12.10%，10.91%，5.86%和 5.08%，平均缩减了 10.23%。说明使用本文算法能有效缩减作业运行时间，提高集群系统性能。

由图 5 可知，运行 TeraSort 工作负载时，使用本文算法相比较其他两种算法，任务的运行时间缩减较为显著。在 5 种不同数据量下，本文算法相较于 FIFO 算法，TeraSort 作业时间分别缩减了 11.11%，6.82%，8.60%，5.53%和 5.20%，平均缩减了 7.45%；本文算法相较于 FAIR 算法，TeraSort 作业时间缩减了 17.24%，9.56%，10.99%，2.38%和 9.20%，平均缩减了 9.87%；说明使用本文算法能有效缩减作业运行时间，提高集群系统性能。

3.3　同种规模数据量不同作业任务实验

实验分别使用 Wordcount，TeraSort 和 *K*-means 三种负载，实验使用的数据量均为 4 GB，使用本文算法相比较其他两种算法的实验结果如图 6 所示。

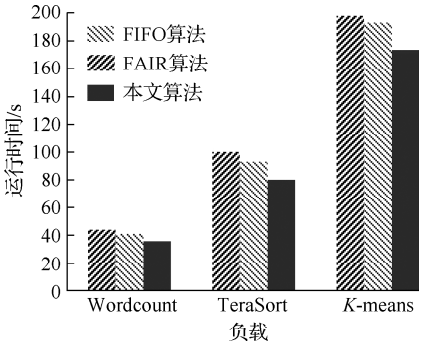

图 6　相同数据量不同负载运行时间对比

Fig.6　Comparison of the running time of the same amount of data with different workloads

由图 6 可知，运行 Wordcount，TeraSort，*K*-means 三种不同工作负载时，使用本文算法相比较其他两种算法，任务的运行时间缩减较为显著。本文算法相较于 FIFO 算法，任务的运行时间分别缩短了 12.29%，14.03%，10.17%，平均缩短了

12.16%；本文算法相较于 FAIR 算法，任务的运行时间分别缩短了 18.49%，19.99%，12.40%，平均缩短了 16.96%。说明使用本文算法能有效提高集群系统运行效率。

4 结 论

为了解决分布式异构集群负载不均衡导致集群运行效率降低的状况，文章对分布式计算框架 Spark 任务调度策略进行优化。考虑集群节点计算能力的差异，将异构集群分成不同性能的区域，利用集群节点静态负载和动态负载的状况优化 Spark 作业任务调度，为作业任务适配相应的工作节点。在获取计算节点动态负载方面，通过 Linux 系统的内存文件系统实现，不会增加异构集群的资源消耗。根据计算节点负载动态调度异构集群计算资源，异构集群运行效率得到有效提升，证明了策略在负载均衡调度上的有效性。

参考文献：

[1] 冯兴杰，贺阳. 基于节点性能的 Hadoop 作业调度算法改进[J]. 计算机应用与软件，2017，34(5)：223-228.

[2] 简琤峰，平靖，张美玉. 面向边缘计算的 Storm 边缘节点调度优化方法[J]. 计算机科学，2020，47(5)：277-283.

[3] ZAHARIA M，CHOWDHURY M，FRANKLIN M J，et al. Spark：cluster computing with working sets[C]//Proceedings of the 2nd USENIX conference on hot topics in cloud computing. New York：ACM，2010：10.

[4] WIKTORSKI T. Data-intensive systems：principles and fundamentals using Hadoop and Spark[M]. Cham：Springer International Publishing，2019.

[5] 郑晓薇，项明，张大为，等. 基于节点能力的 Hadoop 集群任务自适应调度方法[J]. 计算机研究与发展，2014，51(3)：618-626.

[6] XU X L，CAO L L，WANG X H. Adaptive task scheduling strategy based on dynamic workload adjustment for heterogeneous hadoop clusters[J]. IEEE Systems Journal，2016，10(2)：471-482.

[7] YONG M，GAREGRAT N，MOHAN S. Towards a resource aware scheduler in Hadoop[C]//Proc of the 7th IEEE International Conference on Web Services.[S.l.]：IEEE，2009：102-109.

[8] 徐佳俊，刘功申，苏波，等. 基于 Spark 的异构集群调度策略研究[J]. 计算机科学与应用，2016(11)：692-704.

[9] 胡亚红，盛夏，毛家发. 资源不均衡 Spark 环境任务调度优化算法研究[J]. 计算机工程与科学，2020，42(2)：203-209.

[10] CHAMBERS B，ZAHARIA M. Spark：the definitive guide[M]. 张岩峰，王方京，陈晶晶，译. 北京：中国电力出版社，2020.

[11] KOTOULAS S，OREN E，VAN HARMELEN F. Mind the data skew：distributed inferencing by speed dating in elastic regions[C]//Proceedings of the 19th international conference on World wide web. New York：ACM，2010：531-540.

[12] DAVIDSON A，OR A. Optimizing shuffle performance in Spark[EB/OL].[2018-11-25]. https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F13/projects/reports/project16_report.pdf.

[13] 詹剑锋，高婉铃，王磊，等. Big Data Bench：开源的大数据系统评测基准[J]. 计算机学报，2016，39(1)：196-211.

（责任编辑：谭晓荷）