浙江科技学院学报，第 34 卷第 5 期，2022 年 10 月
Journal of Zhejiang University of Science and Technology
Vol. 34 No. 5, Oct. 2022

**doi**：10.3969/j.issn.1671-8798.2022.05.008

# 基于作业类型和优先级权重的容量调度算法

谭　平，徐金美，蒋天煜，Tambominyi Eliasu，丁　进
（浙江科技学院 自动化与电气工程学院，杭州 310023）

**摘　要**：针对 Hadoop 平台默认调度器在异构环境中不能根据作业类型和资源使用情况进行资源调度的问题，进行了 Hadoop 容量调度算法资源调度机制研究，提出一种基于作业类型和优先级权重的容量调度算法。首先通过作业负载分类，将用户提交的作业划分为 CPU（central processing unit，中央处理器）负载类型作业和 I/O（input/output，输入/输出）负载类型作业，并将不同类型作业分配到相应队列中；然后根据权重公式为超过等待时间阈值的作业更新作业优先级，实现队列中的作业排序；最后结合节点实时负载情况优先为优先级高的作业分配资源，从而实现集群性能的提升。结果表明，在异构环境下，相比容量调度算法，本算法将单作业类型和混合作业类型的作业平均完成时间分别缩短了 9.7%和 30.8%，从而有效地提高了作业执行效率和系统的资源利用率。本算法优化了 Hadoop 系统的负载均衡，可为后续相关调度算法的优化研究提供参考。

**关键词**：Hadoop；容量调度器；作业负载分类；节点实时负载；优先级权重
**中图分类号**：TP393.0　　**文献标志码**：A　　**文章编号**：1671-8798(2022)05-0444-08

# Capacity scheduling algorithm based on job type and priority weight

TAN Ping, XU Jinmei, JIANG Tianyu, Tambominyi Eliasu, DING Jin
(School of Automation and Electrical Engineering, Zhejiang University of Science and Technology, Hangzhou 310023, Zhejiang, China)

**Abstract**: Aiming at the problem that the default scheduler of the Hadoop platform fails to carry through resource scheduling based on job types and resource usage in a heterogeneous environment, the resource scheduling mechanism of the capacity scheduling algorithm of the Hadoop platform was studied, proposing a capacity scheduling algorithm based on the job type and priority weight. The algorithm first classified the jobs submitted by users into CPU (central processing unit) load type and I/O (input/output) load type by virtue of workload classification and allocated different types of jobs to the corresponding queues. Then, according to the weight formula, the job priority was updated for the jobs exceeding the waiting time

**收稿日期**：2021-07-13
**基金项目**：国家自然科学基金项目(51677171)；国家重点研发计划项目(2018YFB0606000，2018YFB1702200)
**通信作者**：谭　平(1978—　)，男，江苏省如皋人，教授，博士，主要从事智能安全系统研究。E-mail：115011@zust.edu.cn。

threshold, sequencing jobs in the queues. Finally, combined with the real-time load conditions of the nodes, resources were allocated to the jobs with higher priority, enhancing the cluster performance. The results show that, compared with the capacity scheduling algorithm, the algorithm in this paper shortens the average job completion time of the same type job and the mixed-type job by 9.7% and 30.8%, respectively in a heterogeneous environment, and optimizes the Hadoop system load balancing, and improves the efficiency of job execution and the system resource utilization effectively. This algorithm optimizes the load balancing of the Hadoop system, and provides a reference for the optimization of subsequent relevant scheduling algorithms.

**Keywords**: Hadoop; capacity scheduler; workload classification; real-time load of nodes; priority weight

近年来，云计算已经成为共享资源的一种新方式[1-2]，而 Hadoop 作为主流的云计算开源平台在此背景下得到了广泛的发展和应用。Hadoop 是一个开源且可运行于大规模集群的分布式文件系统和分布式计算框架。调度器作为核心组件直接影响平台性能和资源利用率，其中资源调度算法是提升集群性能的关键。然而默认调度器还存在许多不足[3-5]，其调度算法是基于同构集群的假设构建的，但对实际运用的 Hadoop 集群而言，节点往往是异构的，这种同质性假设会导致异构环境中节点负载不均，且不同类型作业间的资源需求存在差异性，原有调度算法未考虑节点实时负载情况。因此，研究调度算法的改进，对提高 Hadoop 系统资源利用率和性能具有重要的现实意义。

目前 Hadoop 调度算法的研究主要涉及异构环境、智能算法、数据本地性和实时性等方面[6-8]。王溪波等[9]提出一种计算节点负载的任务调度策略，通过节点负载情况减少非本地任务的产生和数据迁移量。潘佳艺等[10]提出负载自适应反馈调度策略，区分了作业类别，解决了负载异构性带来的资源碎片化问题。在此基础上，沈学利等[11]提出基于节点能力的自适应调度算法(node capacity adaptive scheduling，NCAS)，依据节点负载与作业类型分配资源。李曌等[12]提出一种基于作业类别和截止时间的作业调度算法，该算法考虑并优化了异构环境下的作业选择机制。上述研究在异构环境下对调度算法做出改进，但对作业选择机制的优化并不完善，未能满足用户和作业的多样化需求。针对上述调度算法的不足，本研究结合作业负载类型、节点实时负载情况和优先级权重的作业选择机制，提出基于作业类型和优先级权重的容量调度算法。

## 1　Hadoop 调度器及其调度算法

Hadoop1.0 JobTracker(作业管理器)组件任务繁重，负责整个系统的作业调度与资源管理。Hadoop2.0 引入了资源管理框架 YARN(yet another resource negotiator)[13]，减轻了 JobTracker 的任务负担，将资源管理和作业调度分开，由全局资源管理器(resource manager，RM)负责作业的资源分配，应用程序管理器(application master，AM)负责作业管理和监控。分布式文件系统(Hadoop distributed file system，HDFS)[14]和分布式计算框架(map reduce，MR)[15]是 Hadoop 框架核心组成部分，HDFS 为海量数据提供分布式存储服务，MR 为海量数据提供计算。

Hadoop 默认有 3 个调度器[16-17]，分别是先进先出调度器(first in first out scheduler)、容量调度器(capacity scheduler)和公平调度器(fair scheduler)。本研究针对容量调度器提出改进，因此先介绍其资源调度机制。容量调度器支持多队列服务，资源按照一定的权重分配给队列，在其进行资源调度时，先计算队列中运行的任务数和应分得计算资源的比值，选择比值最小的队列执行任务，队列内部再按照初始优先级和先进先出调度机制选择作业执行，在当前运行的作业被终止后，资源才能被分配给新的作业[18]。队列并行执行的任务数等于队列个数。该调度策略的优点是可同时维护多个队列，队列可共享空闲资源，但并未考虑同类型作业之间的资源竞争和节点的实际负载情况，导致集群中节点负载不均；队列内部的作业选择机制过于简单不能满足用户和作业多样化的需求，导致集群资源利用率低下。

## 2 容量调度算法优化与改进

容量调度器按照初始优先级和先进先出调度策略来处理作业，但在实际资源调度时，仅考虑作业初始优先级和作业提交顺序是远远不够的。且不同类型作业之间的资源需求存在着差异性，而容量调度算法并没有对作业进行负载分类。因而相同节点上可能运行着同类型的作业，不能充分利用处理器的性能，从而导致节点上资源相互竞争、资源利用率低下等问题，影响用户作业的处理时间和整个系统的性能。文献[19]针对这一问题设计了一种动态 MR 工作负载预测机制，区分作业负载类型进行任务的调度，本研究以此为基础，结合作业负载类型、节点实时负载情况和作业优先级权重调度提出了基于作业类型和优先级权重的容量调度算法，能自动将用户提交的作业进行作业负载分类并结合节点实时负载情况进行作业调度，能有效地避免同节点的资源竞争问题，通过基于权重作业优先级的作业选择机制满足用户和作业多样化的需求，综合考虑多因素来衡量作业的紧急程度，实现根据用户紧急程度的作业调度，以达到优先分配资源的目的。调度算法流程如图 1 所示，主要分为以下 3 个部分：

1）将用户提交的作业划分为 CPU(central processing unit，中央处理器)负载类型作业和 I/O(input/output，输入/输出)负载类型作业，并将作业自动分配到对应的队列中(内存队列和 I/O 队列)；

2）采集节点资源信息，计算节点负载，按照节点实时负载情况从小到大进行排序，优先给负载小的节点分配任务执行；

3）队列中的作业按照优先级权重选择合适的任务进行调度，作业优先级权重高的作业优先执行。

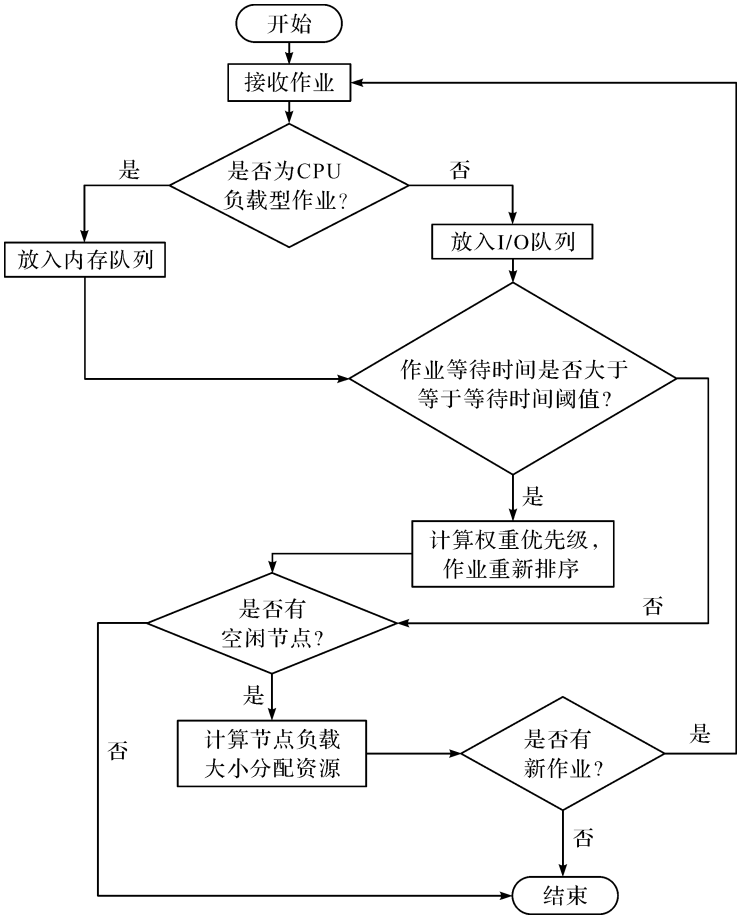


**图 1** 调度算法流程

**Fig. 1** Flow of scheduling algorithm

### 2.1 作业负载分类

MR 程序主要分为两个过程，即映射(map)阶段和化简(reduce)阶段，MR 工作流程如图 2 所示。其

中 map 阶段是 MR 过程的关键，先将输入的数据进行分割，再为每个数据块分配 map 任务。Key/value 键值对作为 map 函数的输入，map 函数输出的中间 key/value 键值对存在本地磁盘中。reduce 阶段运行用户自定义的 reduce 函数对中间 key/value 键值对进行处理。在当前作业的 map 阶段完成之后才会进入 reduce 阶段，且节点执行同一个作业的 map 任务时，每个 map 任务具有相同的运行特性。因此，本研究作业负载分类主要针对 map 阶段来衡量作业的负载情况。根据 CPU 和内存的利用率，对 map 阶段的作业负载进行了分类，划分为 CPU 负载类型作业和 I/O 负载类型作业。

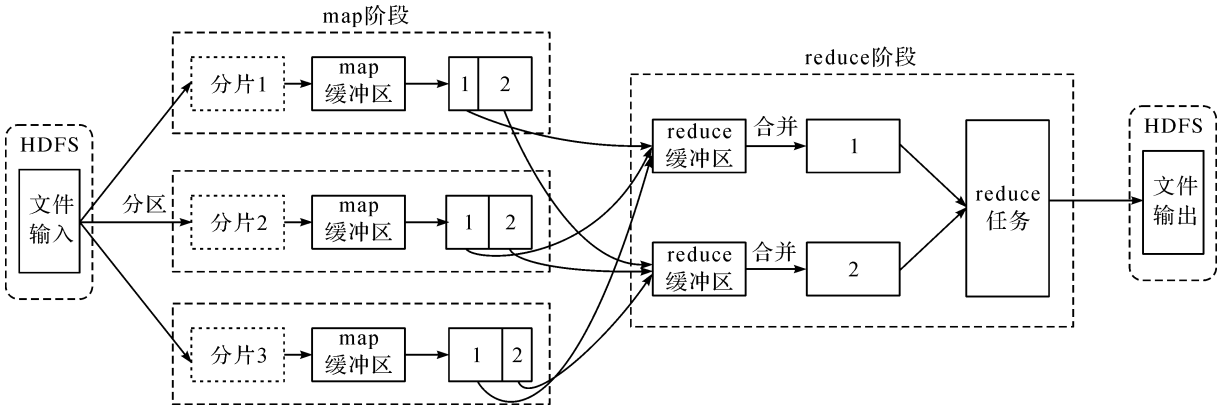


**图 2**　MR 工作流程

**Fig. 2**　Work flow of MapReduce

CPU 负载类型作业指处理速度受 CPU 限制的进程，作业执行时需要进行大量的计算和逻辑判断，消耗 CPU 资源，CPU 占用率高，系统的硬盘/内存效能高于 CPU 效能。I/O 负载类型作业指处理速度受 I/O 子系统限制的进程，作业执行时的状况是 CPU 等待 I/O 的读写，此时系统的 CPU 消耗小、效率高，涉及网络、磁盘 IO 的作业都是 I/O 负载类型作业。

参数 $\beta$ 为 map 任务输出数据 $M_o$ 和输入数据 $M_i$ 的比值，定义如下：

$$\beta=\frac{M_o}{M_i}。\tag{1}$$

该参数反映了 map 任务的负载状况，若作业的 map 任务负载状况相同，即表示具有相同的 $\beta$ 值。由式(1)，可推导出当前该任务在节点上的数据传输速率

$$N_r=\frac{n\times(M_i+M_o)}{T_{map}}=\frac{n\times(1+\beta)\times M_i}{T_{map}}。\tag{2}$$

式(2)中：$n$ 为节点中正在执行的 map 任务数；$T_{map}$ 为 map 任务的完成时间。当节点上的数据传输速率大于磁盘 I/O 速率时，表示当前作业需要进行大量的 I/O 操作，因此本研究将此类作业划分为 I/O 负载类型作业；当一个作业的磁盘 I/O 速率较低时，其 CPU 使用率相对较高，将此类作业划分为 CPU 负载类型作业。I/O 负载类型作业和 CPU 负载类型作业的定义分别如下：

$$\frac{n\times(1+\beta)\times M_i}{T_{map}}>D_r；\tag{3}$$

$$\frac{n\times(1+\beta)\times M_i}{T_{map}}\leqslant D_r。\tag{4}$$

式(3)～(4)中：$D_r$ 为磁盘 I/O 速率。

### 2.2　节点负载排序

在实际集群环境中，Hadoop 大多部署在异构集群上，各个节点计算能力差别较大。容量调度器忽略了节点间的差异，导致节点性能无法最大化。本算法在节点资源被调度之前，实时采集节点资源信息，计算节点实时负载状况，按照节点实时负载情况从小到大进行排序，并且每隔一定的心跳周期更新一次节点队列的节点排序。在对用户提交的作业进行作业负载分类的基础上，对节点负载进行排序，优先分配负载小的节点给作业执行，大大提高作业的执行效率。

节点负载情况受 CPU、内存、磁盘资源、网络状况等因素的影响[20]，本研究中节点负载计算公式如下：

$$w_{\mathrm{CPU}}+w_{\mathrm{mem}}+w_{\mathrm{disk}}+w_{\mathrm{nr}}=1,w_{\mathrm{CPU}},w_{\mathrm{mem}},w_{\mathrm{disk}},w_{\mathrm{nr}}\in[0,1]。\tag{5}$$

$$L_{\mathrm{node}}=w_{\mathrm{CPU}}\times\mu_{\mathrm{CPU}}+w_{\mathrm{mem}}\times\mu_{\mathrm{mem}}+w_{\mathrm{disk}}\times\mu_{\mathrm{disk}}+w_{\mathrm{nr}}\times\mu_{\mathrm{nr}}。\tag{6}$$

式(5)～(6)中：$w_{\mathrm{CPU}}$、$w_{\mathrm{mem}}$、$w_{\mathrm{disk}}$和 $w_{\mathrm{nr}}$分别为节点 CPU、内存、磁盘和网络资源的使用率；$L_{\mathrm{node}}$为节点负载值；$\mu_{\mathrm{CPU}}$、$\mu_{\mathrm{mem}}$、$\mu_{\mathrm{disk}}$和 $\mu_{\mathrm{nr}}$分别为节点 CPU、内存、磁盘和网络资源的权重值。

### 2.3 作业调度执行

容量调度器不支持资源抢占，优先级高的作业会被当前作业阻塞。队列中作业执行顺序会直接影响作业等待时间并间接影响总的作业完成时间。调度算法的核心是如何为作业分配合适的资源，仅考虑作业初始优先级对作业进行调度是不可取的。因此，本算法提出了一种新的作业选择机制，作业的优先级权重能综合考虑作业初始优先级、作业等待时间、作业占用 CPU 和内存资源大小这 4 个因素，根据权重优先级来衡量作业紧急程度，优先执行紧急程度高的作业，以满足用户和作业的多样性需求，从而提高系统性能。

对等待时间超过阈值的作业，通过提高其权重实现作业优先级提升，进而提高作业实时性。对占用 CPU 和内存资源过大的作业，可基于负载类型及节点负载情况进行作业分配优化，降低总的作业完成时间，提高作业执行效率。Hadoop 默认初始作业优先级分为 5 个级别，分别为 VERY_HIGH、HIGH、NORMAL、LOW、VERY_LOW。作业权重优先级能满足用户作业多样性的需求，优先完成紧急程度高的作业。本调度算法在用户提交作业后，首先划分作业的负载类型，并把作业分配到相应的队列中；其次进行作业初始优先级的判断，并将默认的初始作业优先级量化，调度器按照优先级权重公式更新权重优先级，并将队列中的作业重新排序；最后结合节点负载情况优先给权重作业优先级高的作业分配资源。作业优先级权重计算公式如下：

$$P_{\mathrm{w}}=\begin{cases}P,T_{\mathrm{wait}}<T;\\ P+w_{\mathrm{time}}\times\dfrac{T_{\mathrm{wait}}}{T}+w_{\mathrm{CPU}}\times P_{\mathrm{CPU}}+w_{\mathrm{ram}}\times P_{\mathrm{ram}}+w_{\mathrm{priority}}\times P,T_{\mathrm{wait}}\geqslant T。\end{cases}\tag{7}$$

式(7)中：$P_{\mathrm{w}}$ 为权重优先级；$P$ 为作业初始优先级；$T_{\mathrm{wait}}$ 为当前作业等待时间，$T_{\mathrm{wait}}=T_{\mathrm{current}}-T_{\mathrm{start}}$；$T$ 为等待时间阈值；$P_{\mathrm{CPU}}$为当前作业占用 CPU 资源大小；$P_{\mathrm{ram}}$为作业占用内存资源大小；$T_{\mathrm{current}}$为当前时间；$T_{\mathrm{start}}$为作业提交的时间。权重值之和满足

$$w_{\mathrm{time}}+w_{\mathrm{CPU}}+w_{\mathrm{ram}}+w_{\mathrm{priority}}=1,w_{\mathrm{time}},w_{\mathrm{CPU}},w_{\mathrm{ram}},w_{\mathrm{priority}}\in[0,1]。\tag{8}$$

式(8)中：$w_{\mathrm{time}}$、$w_{\mathrm{CPU}}$、$w_{\mathrm{ram}}$和 $w_{\mathrm{priority}}$分别为作业等待时间、作业占用 CPU 资源大小、作业占用内存资源大小和初始优先级的权重值。

## 3 试验结果及性能分析

### 3.1 Hadoop 集群架构

Hadoop 测试集群由 1 台工作站、3 台计算机搭建而成。1 台为主节点，另外 3 台为从节点。4 台主机处于局域网中，网络设备为 1 台小型 8 口 TP-Link 交换机。搭建本集群的运行环境安装版本为：Ubuntu18.04、JDK1.8.0、Hadoop2.7.6，具体集群环境配置见表 1。

**表 1** 集群环境配置

**Table 1** Cluster environment configuration

| 设备 | 节点类型 | IP 地址 | CPU 配置 | 内存/GB |
|---|---|---|---|---|
| 主机 | 主节点 | 172.18.56.218 | Intel E3-1230 | 8 |
| 从机 1 | 从节点 | 172.18.56.195 | Intel i3-3220 | 4 |
| 从机 2 | 从节点 | 172.18.56.29 | Intel i3-4170 | 4 |
| 从机 3 | 从节点 | 172.18.56.3 | Intel i5-8400 | 8 |

试验选取 Hadoop 自带性能测试集：TeraSort 和 GrepCount。文献[19]论证了 TeraSort 为 I/O 负载类型作业，GrepCount 为 CPU 负载类型作业。在本地搭建的 Hadoop 测试集群上分别配置容量调度算法及基于作业类型和优先级权重的容量调度算法，按照相同的方式和作业提交顺序来进行对比试验。试验数据均为重复 5 次测试所取的平均值，以保证其有效性。测试集群网络拓扑图如图 3 所示。

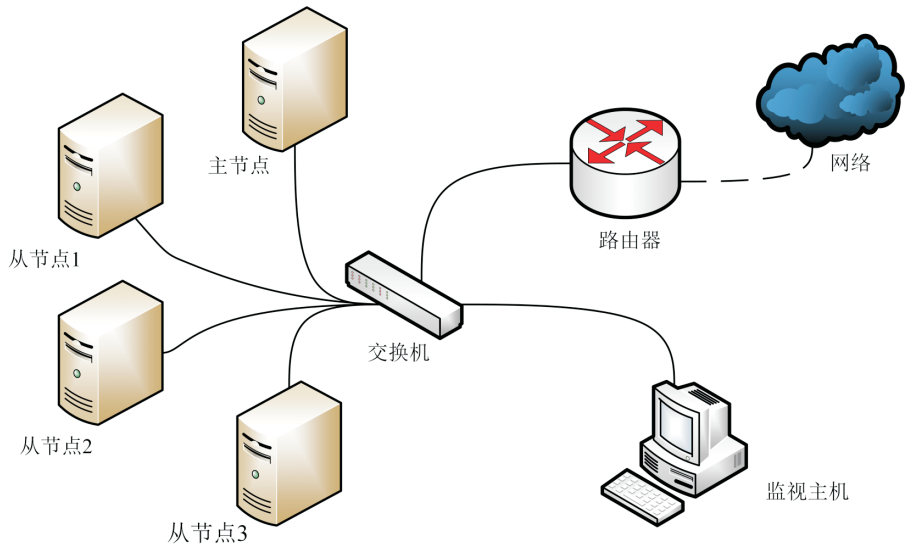


**图 3**　测试集群网络拓扑图

**Fig. 3**　Network topology of test cluster

### 3.2　试验结果与分析

本试验中基于作业类型和优先级权重的容量调度算法等待时间阈值 $T$ 取值为 30 s，权重取值为：$w_{time}=0.2$，$w_{CPU}=0.3$，$w_{ram}=0.3$，$w_{priority}=0.2$。

试验 1 在测试集群上配置和使用容量调度算法及基于作业类型和优先级权重的容量调度算法，运行 TeraSort 和 GrepCount 作业，输入数据量分别为 1、3、5 GB，作业平均完成时间如图 4～5 所示。由图 4～5 可知，本算法明显缩短了 TeraSort 和 GrepCount 的作业平均完成时间，较容量调度算法缩短了 9.7%。本算法对作业进行作业负载分类，虽在一定程度上延长了作业完成时间，但本算法划分了作业负载分类并结合节点实时负载情况将不同类型的作业匹配给适合的节点执行，从而减少了同类作业的资源竞争，提高了资源利用率。

试验 2 依次在测试集群上配置容量调度器和基于作业类型和优先级权重的容量调度器，按照相同的方式和作业顺序提交数据量为 5 GB 的作业流，研究集群运行结果并统计和分析数据。用户提交作业流列表见表 2，作业提交时间间隔为 1 s。

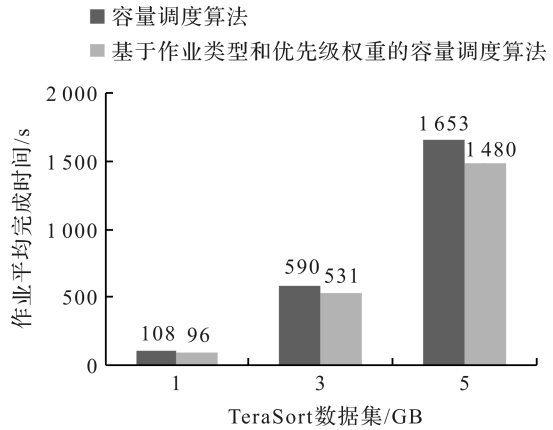


**图 4**　TeraSort 作业平均完成时间

**Fig. 4**　Average job completion time of TeraSort

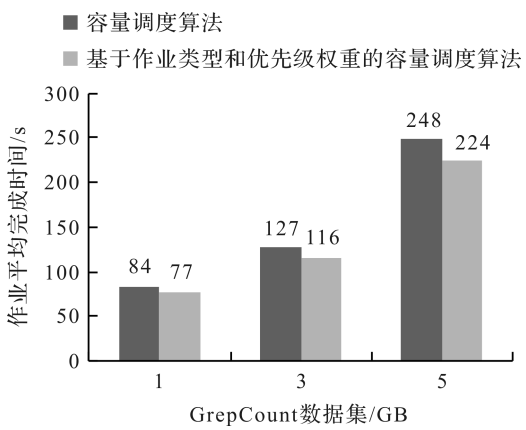


**图 5**　GrepCount 作业平均完成时间

**Fig. 5**　Average job completion time of GrepCount

**表 2** 作业流列表

**Table 2** List of job flow

| 任务名称 | 作业 id | 作业类型 | 初始优先级 | 数据量/MB |
|---|---|---|---|---|
| terasort_1 | T1 | I/O 负载 | VERY_LOW | 100 |
| grepcount_2 | G2 | CPU 负载 | VERY_LOW | 100 |
| terasort_3 | T3 | I/O 负载 | VERY_LOW | 1 024 |
| grepcount_4 | G4 | CPU 负载 | VERY_LOW | 1 024 |
| grepcount_5 | G5 | CPU 负载 | HIGH | 500 |
| terasort_6 | T6 | I/O 负载 | LOW | 100 |
| terasort_7 | T7 | I/O 负载 | VERY_HIGH | 200 |
| grepcount_8 | G8 | CPU 负载 | VERY_HIGH | 500 |
| terasort_9 | T9 | I/O 负载 | NORMAL | 1 024 |
| terasort_10 | T10 | I/O 负载 | HIGH | 500 |

据作业运行日志，统计 5 GB 作业流的队列分配情况，其结果见表 3。本算法配置 2 个队列（内存队列和 I/O 队列）来存放用户提交的作业。由运行日志可以看出调度器区分出了作业负载类型，并准确地将其分配到队列中。

**表 3** 作业队列分配

**Table 3** Job queue allocation

| 任务名称 | 作业 id | 作业类型 | 分配队列 | 作业执行顺序 |
|---|---|---|---|---|
| terasort_1 | T1 | I/O 负载 | I/O 队列 | 1 |
| grepcount_2 | G2 | CPU 负载 | 内存队列 | 2 |
| terasort_3 | T3 | I/O 负载 | I/O 队列 | 10 |
| grepcount_4 | G4 | CPU 负载 | 内存队列 | 6 |
| grepcount_5 | G5 | CPU 负载 | 内存队列 | 5 |
| terasort_6 | T6 | I/O 负载 | I/O 队列 | 9 |
| terasort_7 | T7 | I/O 负载 | I/O 队列 | 3 |
| grepcount_8 | G8 | CPU 负载 | 内存队列 | 4 |
| terasort_9 | T9 | I/O 负载 | I/O 队列 | 8 |
| terasort_10 | T10 | I/O 负载 | I/O 队列 | 7 |

容量调度器作业流测试及基于作业类型和优先级权重的容量调度器作业流测试结果如图 6～7 所示，图中不同颜色代表不同的作业，两幅图中相同颜色代表作业流中的同一作业。结合表 3、图 6 和图 7 可知，本算法改变了作业的执行顺序，且缩短了作业平均完成时间。从单个作业的完成时间来看，大部分作业的执行时间比容量调度器有大幅度的减少。有个别作业如 T3、G4 的完成时间落后于原调度算法，这是因为在执行 T3 作业和 G4 作业之前，用户提交作业的等待时间超过了等待时间阈值，作业根据权重公式更新了优先级和队列中作业执行顺序，优先执行了初始优先级大和需要占用大资源的作业，这些优先执行的作业占用了队列资源，从而拉长了 T3 和 G4 的完成时间。但从整个作业流的完成时间来看，本算法让不同类型作业使用不同的节点资源，有效地将作业流平均完成时间缩短 30.8%，实现了对集群资源的充分利用，提高了集群的吞吐率。

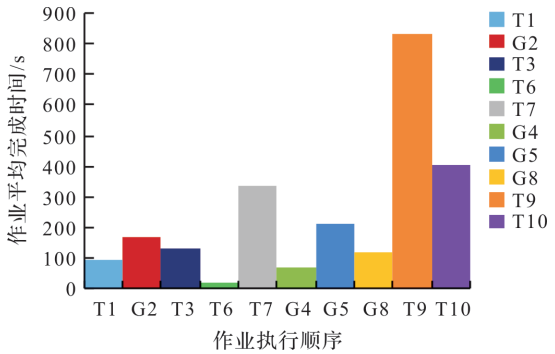


**图 6** 容量调度器作业流测试

**Fig. 6** Job flow test of capacity scheduler

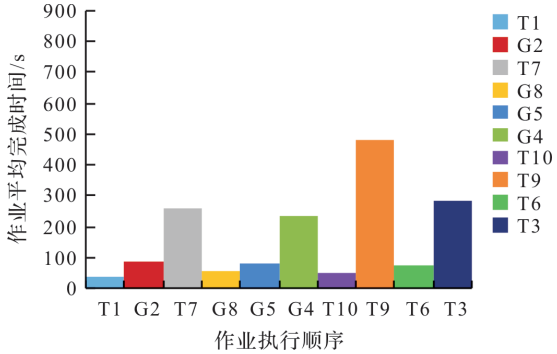


**图 7** 基于作业类型和优先级权重的容量调度器作业流测试

**Fig. 7** Job flow test of capacity scheduler based on job type and priority weight

## 4 结　语

本研究分析了 Hadoop 平台调度算法在异构环境下调度策略的不足，并针对容量调度算法提出了基于作业类型和优先级权重的容量调度算法。在本地搭建集群进行试验，其结果表明：本算法划分了作业负载类型并将其分配到对应的队列中，结合优先级权重对队列中的作业进行重新排序，优先为权重高的作业分配资源，充分利用了集群资源，以缩短了作业平均完成时间。在后续研究中，会考虑延迟作业的备份，对备份任务的处理策略进行研究，进一步优化提高调度算法性能。

**参考文献：**


[1] 段文雪，胡铭，周琼，等. 云计算系统可靠性研究综述[J]. 计算机研究与发展，2020，57(1)：102.

[2] 田倬璟，黄震春，张益农. 云计算环境任务调度方法研究综述[J]. 计算机工程与应用，2021，57(2)：1.

[3] 宁士勇. 虚拟化云计算数据中心资源节能调度算法研究[J]. 计算机应用研究，2021，38(4)：1088.

[4] 王珊珊. 基于 Hadoop 集群作业调度性能优化技术的研究与实现[D]. 沈阳：沈阳工业大学，2020.

[5] VARGA M，PETRESCU-NITA A，POP F. Deadline scheduling algorithm for sustainable computing in Hadoop environment[J]. Computers & Security，2018，76：354.

[6] DU X M，LIU Y，ZHAO C L. A Hadoop Yarn scheduling based on node computing capability and data locality in heterogeneous environments[J]. International Core Journal of Engineering，2018，4(4)：187.

[7] 郑琳，张辉. 云环境下基于群智能算法的大数据聚类挖掘技术[J]. 现代电子技术，2020，43(15)：115.

[8] 董碧莹. 基于 Hadoop 集群作业调度实时性能改进的研究与设计[D]. 沈阳：沈阳工业大学，2019.

[9] 王溪波，王珊珊，王越峰. Hadoop 环境下节点负载均衡调度策略的设计与实现[J]. 电子技术与软件工程，2020，173(3)：183

[10] 潘佳艺，王芳，杨静怡，等. 异构 Hadoop 集群下的负载自适应反馈调度策略[J]. 计算机工程与科学，2017，39(3)：413.

[11] 沈学利，盛方严. 异构资源环境下 Hadoop 节点能力自适应调度算法[J]. 计算机应用研究，2020，37(2)：547.

[12] 李曌，滕飞，李天瑞，等. 一种 Hadoop 中基于作业类别和截止时间的调度算法[J]. 计算机科学，2015，42(6)：28.

[13] KULKARNI A P，KHANDEWAL M. Survey on Hadoop and introduction to YARN[J]. International Journal of Emerging Technology and Advanced Engineering，2014，4(5)：82.

[14] SHVACHKO K，KUANG H，RADIA S，et al. The Hadoop distributed file system[C]//Proceeding of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). Incline Village：IEEE Computer Society，2010：1.

[15] RODRIGUEZ J M，MATEOS C，ZUNINO A. Energy-efficient job stealing for CPU-intensive processing in mobile devices[J]. Computing，2014，96(2)：87.

[16] 廉华，刘瑜. 基于 YARN 资源调度器的 MapReduce 作业数调节方法[J]. 计算机系统应用，2020，29(3)：218.

[17] 郭玉栋，左金平. 基于霍普菲尔德网络的云作业调度算法[J]. 系统仿真学报，2019，31(12)：2859.

[18] JAVANMARDI A K，YAGHOUBYAN S H，BAGHERIFARD K，et al. An architecture for scheduling with the capability of minimum share to heterogeneous Hadoop systems[J]. The Journal of Supercomputing，2020，77：5289.

[19] TAIN C，ZHOU H J，HE Y Q，et al. A dynamic MapReduce scheduler for heterogeneous workloads[C]// Proceeding of the 8th International Conference on Grid and Cooperative Computing. Lanzhou：IEEE Press，2009：218.

[20] 胡丹，于炯，英昌甜，等. Hadoop 平台下改进的 LATE 调度算法[J]. 计算机工程与应用，2014，50(4)：86.