doi:10.11887/j.cn.202206018 http://journal.nudt.edu.cn

节点实时性能自适应的集群资源分配算法*

胡亚红[1],吴寅超[1],朱正东[2]
(1. 浙江工业大学 计算机科学与技术学院(软件学院),浙江 杭州 310023;
2. 西安交通大学 计算机科学与技术学院,陕西 西安 710049)

摘 要:由于配置和所运行作业的不同,集群各节点的实时性能差异较大。为提高集群性能,提出节点实时性能自适应的集群资源分配算法(node real-time performance adaptive cluster resource scheduling algorithm, NPARSA)。节点实时性能用其配置(CPU核数及速度、内存容量、磁盘容量)和实时状态参数(CPU、内存和磁盘的剩余数量及磁盘读写速度)表示。NPARSA根据作业类型自主选择节点性能评价指标的权值,实现节点实时性能对于作业类型的自适应。实时性能最优的节点分配给作业。虚拟机实验和物理集群实验表明,与Spark默认资源分配算法、没有考虑作业类型与节点匹配的算法、使用作业和节点匹配差异程度作为资源分配依据的算法相比,NPARSA能更有效地缩短作业执行时间、提高集群性能。

关键词:资源分配;作业类型;节点实时性能;层次分析法
中图分类号:TP393 **文献标志码**:A **文章编号**:1001-2486(2022)06-144-07

Node real-time performance adaptive cluster resource scheduling algorithm

HU Yahong[1], *WU Yinchao*[1], *ZHU Zhengdong*[2]
(1. College of Computer Science and Technology (College of Software), Zhejiang University of Technology, Hangzhou 310023, China;
2. School of Computer Science and Technology, Xi′an Jiaotong University, Xi′an 710049, China)

Abstract: The real-time performance of each node in one cluster varies greatly due to different configurations and the job running on it. To improve the cluster performance, NPARSA (node real-time performance adaptive cluster resource scheduling algorithm) was proposed. The real-time performance of a cluster node was represented by its configuration (such as the number of its CPU cores, the speed of CPU, memory capacity, and disk capacity) and the real-time state parameters (such as the residual of CPU, memory, and disk). NPARSA chose the attribute weights for a node according to the type of the job to be handled, and assigned nodes with higher priority to the job. Virtual machine experiments and physical cluster experiments prove the effectiveness of NPARSA. Compared with Spark′s default scheduling algorithm, the algorithm that does not consider the job type and node matching, and the algorithm that uses the degree of job and node matching difference as the basis for resource allocation, NPARSA can improve the performance of a cluster and shorten the execution time of user jobs.

Keywords: resource assignment; job type; node real-time performance; analytic hierarchy process

Apache Spark是高性能的大数据处理框架,能够高效地处理批数据和流式数据[1-3]。但Spark默认的资源分配算法仅匹配作业需求和节点的资源情况,不能够保证为用户作业分配到最优资源。资源分配和作业调度属于NP-hard问题[4],是当前的研究热点。综合考虑集群资源的异构性、节点负载的实时变化和用户作业不同特点的资源分配算法,能够有效提高集群性能、缩短作业的完成时间。

为了解决Spark在进行资源分配时未结合节点硬件性能的问题,Xu等提出了一个考虑集群资源属性和作业特性的启发式算法,能够为作业分配到合适的资源[5]。

胡亚红等提出的基于节点优先级的Spark动态自适应调度算法[6](Spark dynamic adaptive scheduling algorithm, SDASA)则是为了综合处理集群的异构性和节点的实时性能。该算法中,节点的性能使用优先级来度量,节点的实时优先级根据节点的资源使用情况等工作状态进行动态计算。

为保证所有作业都能够在截止时间内完成,文献[7]介绍了一个考虑作业价值密度和完成时

* 收稿日期:2021-01-08
基金项目:国家重点研发计划资助项目(2018YFB0204003)
作者简介:胡亚红(1971—),女,陕西西安人,副教授,博士,硕士生导师,E-mail: huyahong@zjut.edu.cn

间限制的硬实时调度算法截止时间及价值密度感知(deadline and value density aware, DVDA)算法[7]。但 DVDA 算法没有考虑集群中节点的实时资源使用状况。

针对电力供应受限的系统,文献[8]分析了常用基于价值的任务调度方法的不足,提出需要在不同的电力供应状态下选择不同的能源分配方案,并在此基础上,提出了基于价值的能耗感知启发式任务调度算法。

为减少存储设备异构的集群中作业的执行时间,文献[9]提出的任务调度策略根据存储类型获得对应存储设备的读取速度,同时考虑数据存储的位置,以完成任务优先级的计算[9]。

充分利用具有较高读写速度的缓存能够缩短数据的读写时间,加快用户作业的完成。文献[10]描述了针对数据密集型任务的并行调度算法。该算法在考虑数据本地性、相关性和负载均衡的条件下,最小化任务的完成时间。此文献对数据密集型负载进行了分析,但没有讨论其他类型负载的情况。

启发式算法是解决调度问题的常用算法,Yu 等提出使用野草算法(invasive weed optimization, IWO)完成异构集群中的任务调度,目标是最小化任务的完成时间。其中,IWO 用于为每一个子作业分配优先级,最早完成时间(earliest finish time, EFT)算法则为子作业分配最优的运行节点[11]。

由于资源分配问题的复杂性,越来越多的研究采用了机器学习的方法。文献[12]讨论了包含 CPU-GPU 的异构集群中多个任务共用 GPU 的任务调度问题。基于 GPU 上任务的细粒度划分和任务间的关系,作者提出了基于深度强化学习和神经协同过滤的两阶段任务调度方法,给各任务分配最合适的节点[12]。

Hu 等在研究中发现,为用户作业分配过多的资源不但会增加资源间的通信开销,使得作业的完成时间(job completion time, JCT)不降反增,而且还造成其他作业无法获得足够资源[13]。为了解决资源过度分配问题,首先利用深度神经网络(deep neural network, DNN)寻找每个作业的最优集群配置;在获得作业最优配置后,使用短作业优化算法为作业预分配集群资源;之后使用启发式算法以平衡多个作业之间的资源分配,从而达到批作业完成时间最短的优化目标。

Spark 默认资源调度方法没有充分考虑计算资源和网络资源之间的均衡。如果集群用于处理网络密集型作业,则可能需要在节点间传输大量的数据,从而导致集群性能下降。针对该问题,Du 等提出了一种任务完成时间感知的作业调度优化方法[14]。该算法首先进行任务完成时间的预测。再利用得到的任务执行时间,从非本地任务调度和网络的传输时间两方面进行优化。

目前常用的集群资源分配算法没有综合考虑节点的实时性能和待运行作业的特点,使得一些节点的性能优势无法完全发挥,而部分作业的执行时间也没有达到最优。下面以 WordCount 负载分别运行于配置不同的三个节点为例说明上述问题。节点配置如表 1 所示。

表 1　WordCount 运行时间对比实验配置

Tab. 1　Setup for WordCount runtime comparison experiments

节点编号	内存/GB	磁盘容量/GB	CPU 核数	CPU 速度/GHz	磁盘类型	负载运行时间/s
1	6	80	2	4	HDD	90.0
2	6	80	2	4	SSD	88.2
3	6	80	2	2	SSD	151.8

节点 1 和节点 2 的区别在于磁盘的性能,节点 2 的 I/O 性能优于节点 1。节点 2 和节点 3 的 CPU 速度不同,节点 2 的 CPU 性能更好。众所周知,WordCount 负载对于节点的 CPU 性能较为敏感。对比相同的 WordCount 负载在 3 个节点的执行时间,可以清楚地看出节点 3 的运行时间比节点 2 的长了 72.11%,主要原因就是节点 3 的 CPU 性能较差。而 WordCount 在节点 1 和 2 的运行时间仅有 2% 的差别,也说明了 WordCount 对节点的 I/O 性能不敏感。因此为用户作业选择适合其特点的节点非常有必要。

目前考虑节点与作业类型匹配的研究不是很多。文献[15]考虑了节点的任务执行特点,使用作业和节点的匹配差异程度作为资源分配的依据。该文中:作业的类型由用户提供,分为计算密集型和内存密集型。节点的负载由其 CPU 分量、内存分量和系统其他分量表示。资源分配的基本思想是将 CPU 利用率低、内存利用率高的节点分配给计算密集型作业;将内存利用率低的节点分配给计算密集型作业。使用该算法时,若是用户对作业不够了解,则其指定的作业类型会不准确。另外,节点负载的影响分量考虑得不是非常全面。它主要考虑的是 CPU 分量、内存分量,其他如磁盘容量等统一归属于其他分量,因而不能对除了

CPU 和内存以外的分量进行更为有效的处理。

本文提出的节点实时性能自适应的集群资源分配算法(node real-time performance adaptive cluster resource scheduling algorithm,NPARSA)旨在综合考虑节点的实时处理性能优先级和用户作业的特点,为作业分配最适合的节点,从而缩短作业运行时间,提升集群的性能。

1 节点性能评价及资源调度算法

NPARSA 根据用户作业的类型自适应地进行节点优先级评价指标权值的选择,从而计算出各个节点处理此作业的性能优先级。

1.1 用户作业类型的判定

CPU 密集型作业和内存密集型作业是常见类型的用户作业,它们对于集群有着不同的资源需求。作业的类型可以使用运行作业需要的 CPU 核数和内存数量进行判定,式(1)给出了作业类型的判定方法。

$$J = \frac{M_r}{C_r} \tag{1}$$

作业的类型使用变量 J 进行判定。M_r 是用户要求为作业提供的内存数量,而用户要求提供给作业的 CPU 核数用变量 C_r 表示。为了选择合理的 J 阈值以区分作业类型,本文进行了大量的实验。实验分别使用 Spark 默认调度算法和 NPARSA 运行相同的作业,计算出在不同 J 阈值下 NPARSA 能够产生的系统性能提升百分比,实验结果见表 2。

表 2 不同 J 阈值下集群性能提升效果

Tab. 2 Cluster performance improvement under different thresholds of J

J 阈值	集群性能提升率/%
0.8	-4.27
0.9	-0.81
1.0	0.00
1.1	6.43
1.2	3.27
1.3	3.01
1.4	-1.20

对表 2 进行分析发现,当 J 阈值为 1.1 时,NPARSA 能够得到最好的效果。因此,将作业类型 J 的阈值定为 1.1。当一个作业的 J 值大于 1.1 时,判定该作业为内存密集型,否则判定为 CPU 密集型作业。NPARSA 将根据作业的类型自适应地选择节点优先级计算时需要的权值。

1.2 节点性能优先级的评价

异构集群中的节点性能差别较大,有些节点适合处理 CPU 密集型作业,有些则适合运行内存密集型作业。因此根据当前作业的类型自适应地分析节点性能很有必要。

节点性能 P 由其静态性能指标和动态性能指标表示,可以使用式(2)计算得到。

$$P = \alpha S + \beta D \tag{2}$$

式中:S 表示节点的静态性能指标;D 表示节点的动态性能指标;α 和 β 为对应的权值,且 $\alpha + \beta = 1$。

节点的静态性能评价指标和动态性能评价指标分别使用式(3)和式(4)计算。静态性能指标包括其 CPU 核数(C_s)、内存数量(M_s)、磁盘大小(S_{ts})和 CPU 速度(S_{ps})。动态性能评价指标为作业运行中节点实时的 CPU 剩余率(C_d)、内存剩余率(M_d)、磁盘读写速度(S_{pd})和磁盘剩余率(S_{td})等。

$$S = \alpha_1 C_s + \alpha_2 M_s + \alpha_3 S_{ts} + \alpha_4 S_{ps} \tag{3}$$

$$D = \beta_1 C_d + \beta_2 M_d + \beta_3 S_{pd} + \beta_4 S_{td} \tag{4}$$

其中:节点静态和动态性能各评价指标的权值 $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$,$\beta_1 + \beta_2 + \beta_3 + \beta_4 = 1$。

1.3 节点性能优先级影响因素权值的计算

层次分析法(analytic hierarchy process,AHP)是将定量分析和定性分析结合的方法,常用于权值计算。使用 AHP 确定影响节点性能优先级各因素的权值,图 1 给出了节点性能优先级评价指标体系的层次结构模型。请专家针对 CPU 密集型和内存密集型作业的特点,分别对各影响因素进行评分。这样,NPARSA 就能够根据当前需要处理作业的类型自适应地进行节点实时性能优先级的计算。

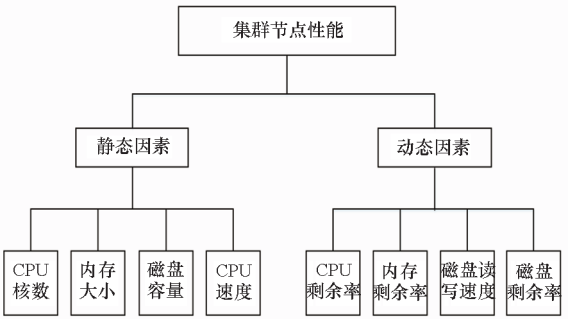

图 1 节点性能优先级评价指标体系层次结构模型

Fig. 1 Hierarchical structure model for the node performance evaluation index system

1.3.1　针对 CPU 密集型作业

经专家打分,静态因素的四个影响因素的判断矩阵是:

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{6} & 2 \\ 2 & 1 & \frac{1}{4} & 2 \\ 6 & 4 & 1 & 8 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{8} & 1 \end{bmatrix}$$

静态因素权值向量 $\boldsymbol{W}=[0.113, 0.641, 0.073, 0.173]^{\mathrm{T}}$,计算得到对应的一致性比率(consistency ratio, CR)的取值为 0.017,通过一致性检验。因此各个静态因素的权值如下:CPU 核数为 0.173、内存大小为 0.641、磁盘容量为 0.113、CPU 速度为 0.073。

动态因素的四个影响因素经专家打分,得到的判断矩阵为:

$$\begin{bmatrix} 1 & \frac{1}{3} & 2 & \frac{1}{2} \\ 3 & 1 & 6 & 1 \\ \frac{1}{2} & \frac{1}{6} & 1 & \frac{1}{4} \\ 2 & 1 & 4 & 1 \end{bmatrix}$$

动态因素权值向量 $\boldsymbol{W}=[0.344, 0.422, 0.156, 0.078]^{\mathrm{T}}$,计算得到 CR 为 0.008,也通过一致性检验。则各个动态因素的权值如下:CPU 剩余率为 0.422、内存剩余率为 0.344、磁盘读写速度为 0.156、磁盘剩余率为 0.078。

式(2)中的静态性能指标和动态性能指标对应的判断矩阵为:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

因此 $\alpha=0.5, \beta=0.5$。

1.3.2　针对内存密集型作业

经专家打分,静态因素的四个影响因素的判断矩阵是:

$$\begin{bmatrix} 1 & 2 & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{6} & \frac{1}{4} \\ 4 & 6 & 1 & 2 \\ 2 & 4 & \frac{1}{2} & 1 \end{bmatrix}$$

静态因素权值向量 $\boldsymbol{W}=[0.138, 0.513, 0.275, 0.074]^{\mathrm{T}}$, CR 为 0.003 9,通过一致性检验。各个静态因素的权值如下:CPU 核数为 0.138、内存大小为 0.513、磁盘容量为 0.275、CPU 速度为 0.074。

动态因素的四个影响因素经专家打分,构造出的判断矩阵为:

$$\begin{bmatrix} 1 & \frac{1}{4} & \frac{1}{2} & \frac{1}{2} \\ 4 & 1 & 2 & 2 \\ 2 & \frac{1}{2} & 1 & \frac{1}{2} \\ 2 & \frac{1}{2} & 2 & 1 \end{bmatrix}$$

动态因素权值向量 $\boldsymbol{W}=[0.110, 0.442, 0.262, 0.186]^{\mathrm{T}}$, CR 值为 0.023,通过一致性检验。各个动态因素的权值如下:CPU 剩余率为 0.110、内存剩余率为 0.442、磁盘读写速度为 0.262、磁盘剩余率为 0.186。

α, β 取值与 CPU 密集型作业的计算方法相同,均为 0.5。

当需要处理的用户作业为 CPU 密集型时,NPARSA 使用 1.3.1 节给出的动静态因素对应的权值;而当需要处理的用户作业为内存密集型时,NPARSA 则会自适应地选择 1.3.2 节给出的动静态因素对应的权值。

1.4　算法描述

NPARSA 运行于 Spark 系统中的 Master 节点。各个 worker 节点实时采集自身的状态信息,Master 则利用 Spark 的心跳机制进行信息的收集,完成各节点实时优先级的计算。算法 1 给出了 NPARSA 的实现。

算法 1　节点性能自适应的资源调度算法

Alg. 1　Node performance adaptive resource scheduling algorithm

输入:集群的用户作业集合 A 和集群中的节点集合 N

输出:为每一个作业分配的节点

1. $i=1$
2. 使用式(1)计算集合 A 中作业 A_i 的类型
3. 根据作业类型选择节点优先级权值,并使用式(2)~(4)计算集合 N 中各个节点的实时性能
4. 将节点的性能作为节点优先级,按照优先级大小将满足作业 A_i 资源要求的节点分配给 A_i
5. $i:=i+1$
6. 如果已经处理了作业队列中所有的作业,算法结束;否则转第 3 步

2　虚拟机实验

首先在虚拟机上运行 NPARSA,以考察其有效性。

2.1 实验环境和数据集

实验中共使用四个虚拟机节点，分别作为 Spark 系统中的主节点和从节点，节点配置见表 3。

表 3 节点配置

Tab. 3 Nodes configuration

节点	CPU 主频/GHz	CPU 核数	内存/GB	硬盘容量/GB	硬盘类型
主节点	3.2	2	5	50	SSD
从节点 1	3.2	2	4	50	SSD
从节点 2	3.2	1	4	50	SSD
从节点 3	3.2	2	4	50	HDD

为了构建异构环境，各节点的内存容量、CPU 核数或磁盘的读写速度有所不同。其中主节点的内存较大；从节点 2 的 CPU 仅有 1 核。为了使节点的 I/O 性能有一定的差异，从节点 3 的存储采用了 HDD，而其他节点硬盘为 SSD。

本文采用的实验数据来源于 BigDataBench[16]，选用了 WordCount、Sort、*K*-means 和 PageRank 四种常用的 CPU 密集型和内存密集型负载。

2.2 实验结果及分析

2.2.1 单作业实验

本实验的工作负载为 WordCount、Sort、*K*-means 和 PageRank。其中 WordCount、*K*-means 和 PageRank 要求 CPU 核数为 4，内存为 4 GB，属于 CPU 密集型负载；Sort 要求 CPU 核数为 4，内存为 8 GB，是内存密集型负载。

WordCount 和 Sort 实验中，Spark 的资源调度算法分别使用 Spark 默认调度算法、SDASA[6]、文献[15]描述的算法（为方便描述，称为 Difference 算法）和 NPARSA。实验结果如图 2 和图 3 所示。

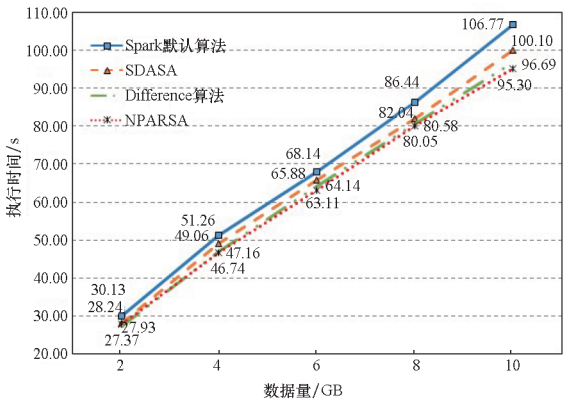

图 2 WordCount 作业完成时间比较

Fig. 2 Comparison of the completion time of WordCount

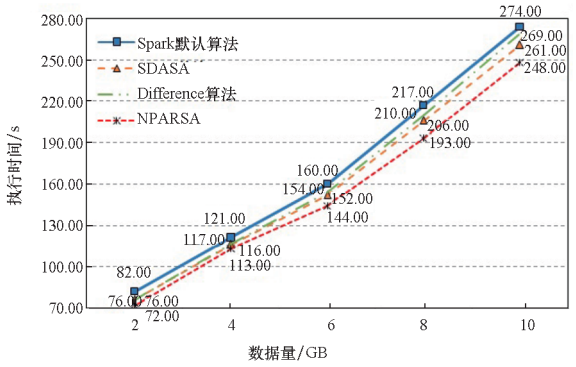

图 3 Sort 作业完成时间比较

Fig. 3 Comparison of the completion time of Sort

从图 2 可以看出，与 Spark 默认调度算法相比，NPARSA 将 WordCount 作业时间平均缩短了 8.33%。与 NPARSA 动态调整节点优先级相同，SDASA 也是结合节点的资源状态，实时调整集群中各个节点的优先级，但是 SDASA 没有考虑用户作业类型。与 SDASA 相比，NPARSA 将系统性能平均提升了 3.45%。与 Difference 算法进行对比，当 WordCount 数据量为 2 GB 时，NPARSA 性能略低于 Difference 算法。但随着数据量的增加，NPARSA 有着更优的表现，系统性能平均提升 0.51%。

从图 3 可以看出，当运行 Sort 作业时，NPARSA 的表现优于所有的对比算法。与 Spark 默认算法、SDASA 和 Difference 算法相比，使用 NPARSA 的作业执行时间平均缩短了 9.87%、4.88% 和 6.22%。

针对 *K*-means 和 PageRank 负载，实验分别在使用 Spark 默认调度算法、SDASA 和 NPARSA 的 Spark 集群中运行。*K*-means 负载的实验结果如图 4 所示。

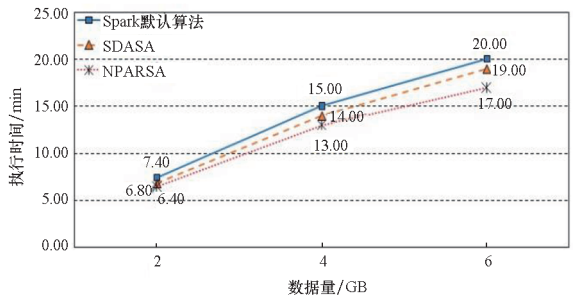

图 4 *K*-means 作业完成时间比较

Fig. 4 Comparison of the completion time of *K*-means

从图 4 可以看出，执行 *K*-means 作业时，与 Spark 默认算法相比，NPARSA 将作业的执行效率平均提升 13.95%；和 SDASA 相比，平均提升 7.85%。

图 5 给出了对 PageRank 负载进行的实验结果。实验中数据集的 Page 数目分别为 2^{60}、2^{70}、2^{80}、2^{90}和 2^{100}。实验结果表明,相对于 Spark 默认算法和 SDASA,NPARSA 完成作业的时间平均缩短了 17.46% 和 6.75%。

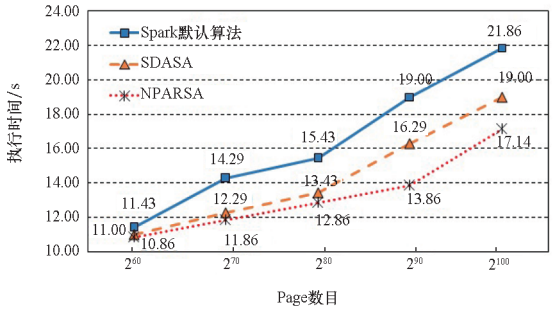

图 5　PageRank 作业完成时间比较

Fig. 5　Comparison of the completion time of PageRank

从图 2 ~ 5 不难看出,无论是运行 WordCount、*K*-means、PageRank 这样的 CPU 密集型作业,还是 Sort 这类内存密集型作业,NPARSA 的表现都较为优秀。并且随着作业数据量的增大,使用 NPARSA 能够更加有效地缩短作业的执行时间,提升集群效率。

2.2.2　多作业并行实验

为考察当多个作业并行运算时 NPARSA 的性能,进行了 WordCount 和 Sort 两类任务的并行运行实验,图 6 给出了实验结果。

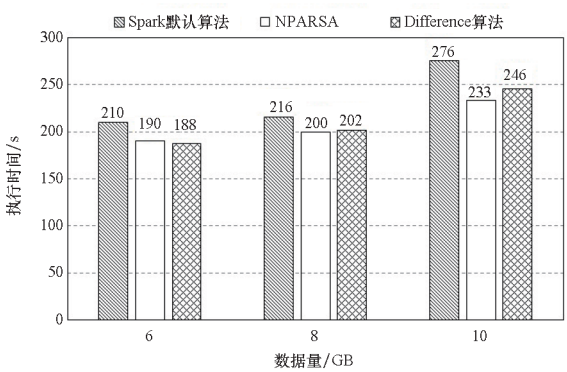

图 6　并行运行的作业执行时间对比

Fig. 6　Completion time comparison for parallel jobs

可以看到,当多个负载并行执行时,与 Spark 默认调度算法相比,NPARSA 将作业的平均执行时间缩短了 10.84%;与 Difference 算法相比,作业平均执行时间也减少了 1.74%。

从上述实验结果可以看出,因为节点的实时性能是根据所要处理作业的类型计算的,NPARSA 能够给每个作业分配合适的集群资源,从而提高了集群的性能,缩短了作业的完成时间。

2.2.3　不同节点数目实验

为了测试集群中节点的数目对于 NPARSA 性能的影响,本节修改了虚拟机实验中的节点数目,删除了原四节点集群中的 Slave1,构建的新集群中仅包含三个节点。在此三节点集群上使用 2.2.1 节描述的方式运行 PageRank,得到的实验结果如图 7 所示。

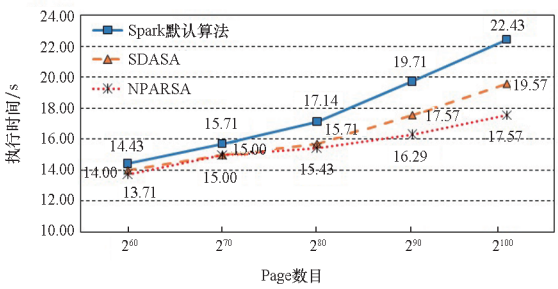

图 7　三节点集群中 PageRank 作业完成时间比较

Fig. 7　Comparison of the completion time of PageRank in the cluster with 3 nodes

通过分析计算,在三节点集群中,NPARSA 完成 PageRank 作业的时间比 Spark 默认算法和 SDASA 平均缩短 11.71% 和 4.28%。在四节点集群中,完成同样数量的 PageRank 作业,NPARSA 完成作业的时间比 Spark 默认算法和 SDASA 平均缩短 17.46% 和 6.75%。因此可以看出,随着集群中节点数目的增加,NPARSA 的优势可以得到更好的体现。

3　小规模物理集群实验

为了避免虚拟机实验的偏差,进一步验证 NPARSA 的有效性,搭建了一个小规模的物理集群,完成了 PageRank 负载的对比实验。集群中有三个节点,其中一个是主节点,两个从节点,节点的配置如表 4 所示。为了使集群具有异构性,三个节点在 CPU 速度、核数、内存大小和磁盘读写速度上配置不同。

表 4　物理集群实验节点配置

Tab. 4　Configurations of the nodes for the physical cluster experiments

节点	CPU 主频/GHz	CPU 核数	内存/GB	硬盘容量/GB	硬盘类型
主节点	2.8	6	12	100	SSD
从节点 1	2.4	4	4	100	HDD
从节点 2	2.8	2	8	100	SSD

实验中 PageRank 的 Page 数目分别为 2^{60}、

2^{70}、2^{80}、2^{90}、2^{100}和2^{110}。实验运行7次,取各次运行结果的平均值作为实验结果,如图8所示。从图8中可以看出,在物理集群实验中,NPARSA仍能比Spark默认调度算法和SDASA有更为良好的表现,其完成PageRank作业的平均时间比Spark默认算法减少了37.39%,比SDASA减少了11.93%。

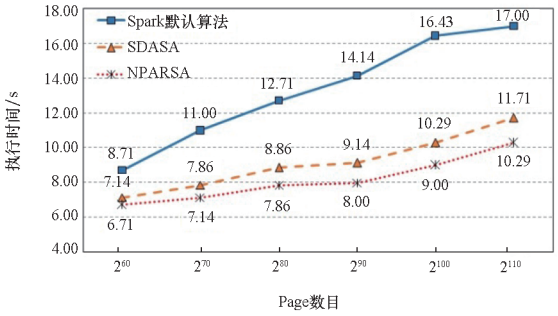

图8 物理集群中PageRank作业完成时间比较

Fig. 8 Comparison of the completion time of PageRank running in a physical cluster

4 结论

为了给每一个用户作业分配最适合其特点的节点,本文提出了NPARSA。NPARSA在计算节点的实时优先级时,会根据当前作业的类型自适应地选择节点优先级评价体系中各指标的权值。虚拟机实验和小规模的物理集群实验结果均表明,与Spark默认调度算法、SDASA和Difference算法相比,NPARSA能够减少用户作业的执行时间。

为了进一步优化NPARSA并验证其有效性,后续的研究内容包括:

1)采用深度学习的方法对更多种用户作业的类型,如I/O密集型、网络密集型等进行准确判断,从而使算法可以更好地为更多类型的作业进行有效的资源分配。

2)增加节点静态和动态资源影响因素,如CPU的最大频率、内存的带宽等,以便更加全面准确地衡量节点的实时性能。

3)在大规模的物理集群上进行实验,进一步验证算法的可扩展性。

参考文献(References)

[1] 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, 2010.

[2] Anon. Apache Spark [CP/OL]. (2009-01-01)[2021-01-01]. http://spark.apache.org/.

[3] 廖湖声, 黄珊珊, 徐俊刚, 等. Spark性能优化技术研究综述[J]. 计算机科学, 2018, 45(7): 7-15, 37.
LIAO H S, HUANG S S, XU J G, et al. Survey on performance optimization technologies for spark[J]. Computer Science, 2018, 45(7): 7-15, 37. (in Chinese)

[4] SUKHOROSLOV O, NAZARENKO A, ALEKSANDROV R. An experimental study of scheduling algorithms for many-task applications[J]. The Journal of Supercomputing, 2019, 75: 7857-7871.

[5] XU L N, BUTT A R, LIM S H, et al. A heterogeneity-aware task scheduler for spark [C]//Proceedings of IEEE International Conference on Cluster Computing, 2018.

[6] 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.
HU Y H, SHENG X, MAO J F. Task scheduling optimization in Spark environment with unbalanced resources [J]. Computer Engineering & Science, 2020, 42(2): 203-209. (in Chinese)

[7] WANG G L, XU J G, LIU R F, et al. A hard real-time scheduler for Spark on YARN [C]//Proceedings of 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2018.

[8] KUMBHARE N, MARATHE A, AKOGLU A, et al. A value-oriented job scheduling approach for power-constrained and oversubscribed HPC systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(6): 1419-1433.

[9] PAN F F, XIONG J, SHEN Y J, et al. H-scheduler: storage-aware task scheduling for heterogeneous-storage spark clusters[C]// Proceedings of IEEE 24th International Conference on Parallel and Distributed Systems, 2019.

[10] MENG X, GOLAB L. Parallel scheduling of data-intensive tasks[C]//Proceedings of European Conference on Parallel Processing, 2020.

[11] YU S Y, LI K L, XU Y M. A DAG task scheduling scheme on heterogeneous cluster systems using discrete IWO algorithm[J]. Journal of Computational Science, 2018, 26: 307-317.

[12] ZHANG H T, GENG X, MA H D. Learning-driven interference-aware workload parallelization for streaming applications in heterogeneous cluster[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(1): 1-15.

[13] HU Z Y, LI D S, GUO D K. Balance resource allocation for Spark jobs based on prediction of the optimal resource[J]. Tsinghua Science and Technology, 2020, 25(4): 487-497.

[14] DU H Z, ZHANG K K, HUANG S. OctopusKing: a TCT-aware task scheduling on spark platform[C]//Proceedings of IEEE 25th International Conference on Parallel and Distributed Systems, 2019.

[15] 韩庆亮. 基于LSTM神经网络的Hadoop集群节能问题研究[D]. 青岛: 山东科技大学, 2018.
HAN Q L. Research on energy saving of hadoop cluster based on neural network LSTM[D]. Qingdao: Shandong University of Science and Technology, 2018. (in Chinese)

[16] 詹剑锋, 高婉铃, 王磊, 等. BigDataBench: 开源的大数据系统评测基准[J]. 计算机学报, 2016, 39(1): 196-211.
ZHAN J F, GAO W L, WANG L, et al. BigDataBench: an open-source big data benchmark suite[J]. Chinese Journal of Computers, 2016, 39(1): 196-211. (in Chinese)