·先进计算·　　文章编号：1000-3428(2016)01-0031-05　　文献标识码：A　　中图分类号：TP301

异构 Spark 集群下自适应任务调度策略

杨志伟，郑　烃，王　嵩，杨　坚，周乐乐

（中国科学技术大学自动化系，合肥 230027）

摘　要： Spark 是一种基于内存的类 Hadoop MapReduce 高效大数据处理平台，但其默认的任务调度策略在异构 Spark 集群下未考虑到节点的能力差异，降低了系统性能。为此，提出一种基于异构 Spark 集群的自适应任务调度策略。该策略通过监测节点的负载及资源利用率，分析监测得到的参数，自适应动态调整节点任务分配权值。实验结果表明，在异构节点情况下，该策略在作业完成时间、节点工作状态及资源利用率方面的性能均优于默认的任务调度策略。

关键词： Spark 平台；异构集群；自适应；任务调度；监测；权值

中文引用格式： 杨志伟，郑　烃，王　嵩，等. 异构 Spark 集群下自适应任务调度策略[J]. 计算机工程，2016，42(1)：31-35，40.

英文引用格式： Yang Zhiwei，Zheng Quan，Wang Song，et al. Adaptive Task Scheduling Strategy for Heterogeneous Spark Cluster[J]. Computer Engineering，2016，42(1)：31-35，40.

Adaptive Task Scheduling Strategy for Heterogeneous Spark Cluster

YANG Zhiwei，ZHENG Quan，WANG Song，YANG Jian，ZHOU Lele

（Department of Automation，University of Science and Technology of China，Hefei 230027，China）

【Abstract】 Spark is a kind of efficient big data processing platform based on memory and similar to Hadoop MapReduce. But the Spark default task scheduling strategy does not take the different capacity of node into account for heterogeneous Spark cluster，thus leading to the low system performance. For this problem，this paper presents an adaptive task scheduling strategy for heterogeneous Spark cluster，which analyzes parameters from surveillance to dynamically adjust the task allocation weights of nodes through monitoring the load and resource utilization of nodes. Experimental result validates that this strategy for heterogeneous nodes is superior to the default task scheduling strategy in aspects like task completion time，nodes working state and resource utilization.

【Key words】 Spark platform；heterogeneous cluster；adaptive；task scheduling；monitoring；weight

DOI： 10.3969/j.issn.1000-3428.2016.01.006

1　概述

近年来，大数据的普及使得大数据计算平台 Spark[1] 备受关注。Spark 是提出的继 Hadoop[2] 后的一种基于内存的快速通用可扩展数据分析引擎，是类 Hadoop MapReduce 的通用并行计算框架。与 Hadoop MapReduce 相比，Spark 基于内存的运算比其快 100 倍，而基于硬盘的运算也要快 10 倍。

Spark 中的任务调度功能就是将作业划分为多个任务，由作业服务器分配到不同的任务服务器上执行。随着时间的推移，企业数据中心硬件资源不断更新换代，Spark 集群很容易产生异构性。由于异构集群下各节点计算能力不一致，因此同一任务分配在不同节点上将会对节点负载造成不同的影响。默认的 Spark 任务调度方式没有考虑集群异构性[3-5] 以及节点当前的资源剩余情况。

针对分布式集群异构的情况下，如何提高系统性能的任务调度问题，文献[4]提出了一种推断式任务执行策略(STES)：如果一个集群已经完成了大部分任务，少部分任务由于分配在较差的硬件配置机器上还未完成，则启动多个未完成任务的副本并行执行，从而减少整个作业的完成时间。文献[5]提出

基金项目： 国家自然科学基金资助项目(61174062)。

作者简介： 杨志伟(1989－)，男，硕士研究生，主研方向为任务调度策略、网络传播与控制；郑　烃，副教授、博士；王　嵩，讲师、博士；杨　坚，副教授、博士生导师；周乐乐，硕士研究生。

收稿日期： 2015-01-09　　**修回日期：** 2015-02-07　**E-mail：** yzw1989020923@163.com

了一种任务节点资源感知策略(TRAS):每个任务节点收集自身资源消耗信息汇报给控制节点,用于服务之后的任务调度。由于采用了相应的自适应调度策略,上述算法均取得了可观的性能提升,但只适用于 Hadoop 平台,不适用于 Spark 系统。因为 Hadoop 平台中任务分配是采用请求方式,即任务执行节点向任务调度节点请求任务,任务调度节点再为其分配任务。而 Spark 系统中是任务调度节点主动分配任务到任务执行节点,因此,本文提出一种能够考虑集群节点异构性和节点资源动态性的 Spark 集群自适应任务调度策略 HSATS。

2 研究背景

2.1 Spark 平台架构

Spark 集群主要包括 Master 节点与 Worker 节点。Master 节点主要负责管理集群,而 Worker 节点主要负责任务的执行。目前 Spark 支持不同的部署方式,包括伪分布、Standalone,Mesos,Yarn 等模式,采用了具有粗颗粒度特性的集合 RDD(Resilient Distributed Datasets)[6-7],并且提供丰富的 Scala,Java,Python 接口语言及交互式 Shell 来提高其易用性。因为 Spark 是基于内存的迭代计算框架,所以非常适合需要多次操作特定数据集的应用场景。Spark 提供的数据集操作类型有很多种,主要分为变换(transformation)与动作(action)[8-9]。应用(application)在提交后,不同的 action 将为作业划分不同的 job,每个 job 又将根据实际情况划分为不同 stage,而不同 stage 中包括一个 task 的集合,再根据任务调度器策略,分配 task 到指定节点上执行。如无特殊说明,本文假设任务调度由 Master 节点控制。

2.2 Spark 任务调度策略

当 Spark 系统同时执行多个任务时,任务调度需要将多个任务分配到合适的节点上执行。Spark 中默认的任务调度策略如下:

(1)Spark 任务调度器从注册的 Worker 节点中随机选取部分节点。

(2)遍历第一步挑选的节点,将任务分配给本地化[10]最高的任务执行节点。

2.3 Spark 存在的问题

上述 Spark 任务调度策略假设集群节点为同构,即节点资源基本一致。但是在数据中心等部署场景中,随着时间的推移以及业务量的增加,集群极有可能产生较大的异构性。与此同时,每个节点的负载及资源利用率是动态变化的。在异构 Spark 集群下,每个节点的当前计算能力是非常不一致的,这种调度策略会带来以下 2 个问题:

(1)节点计算任务分配的相对不均衡性,即:高配置的节点经常处于饥饿状态,而配置较低的节点经常处于满负荷状态。

(2)系统总体计算能力受限和总体任务完成时间较长,性能较低。

针对以上问题,本文提出异构 Spark 集群下自适应任务调度策略 HSATS。

3 HSATS 调度策略

3.1 HSATS 调度设计思想与系统架构

考虑到集群异构节点的情况,HSATS 策略的主要设计思想如下:(1)在一个运行中的 Spark 集群中,每个 Worker 节点可以根据自身资源情况及负载变化情况,定期动态地调节其被选择执行任务的权值。选择 CPU 利用率、内存利用率、单核平均队列长度作为衡量节点负载及资源情况,从而计算权值的指标[3]。(2)Master 节点进行任务调度时,读取各节点权值,优先选择权值较大节点。具体过程如图 1 所示。

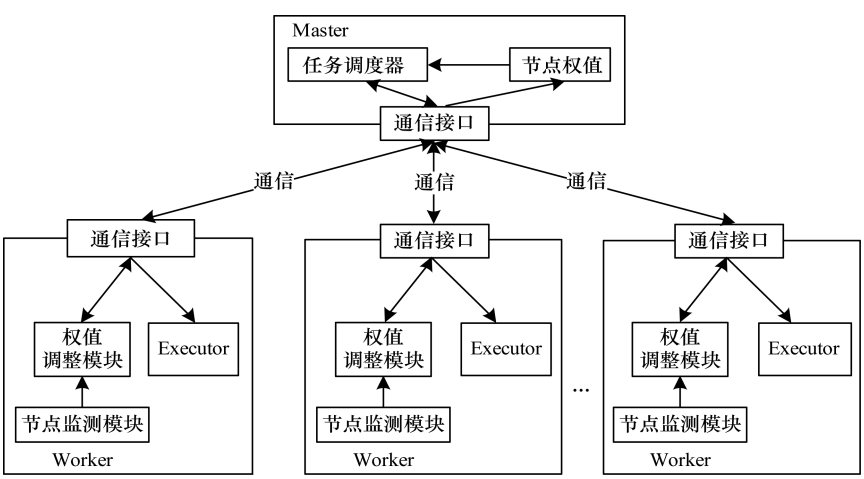

图 1 HSATS 任务调度框架

每个 Worker 节点通过节点监测模块来定期检测自身负载及资源情况,汇报给权值调整模块,权值调整模块确定在一个长度 N 的周期内,是否需要调整其权值。当 Master 节点需要进行任务调度时,首先读取 Worker 节点的当前权值,优先选择权值较大的部分节点进行任务分配。节点间通信采用 akka[11] 进行。

3.2　HSATS 描述

3.2.1　定义

定义 1(节点权值)　定义 $Weight_i(i=1,2,\cdots,M)$ 表示节点权值,其中,M 表示节点个数。每个 Worker 节点均有一个权值,根据自身负载及资源情况动态调整,表示其被选择中执行任务的概率大小。值越大,表示当前节点能力越强,被选择中的概率越大。

定义 2(资源利用率衡量指标)　选择 CPU 利用率 CU、内存利用率 MU 及单核平均队列长度 SCL 作为衡量节点负载及资源情况指标,并且 C_{thre},M_{thre},L_{thre} 分别表示其对应的阈值。

CU 是通过连续 2 次采样值的差值得到,定义如下:

$$CU_k=\frac{CPUtime_{k_2}-CPUtime_{k_1}}{CPUTotal_{k_2}-CPUTotal_{k_1}},k=1,2,\cdots \quad (1)$$

其中,CU_k 表示第 k 次实时监测时 CPU 利用率;$CPUtime$ 表示用户模式、低优先级用户模式、系统模式时间之和,下标 k_1 表示第 k 次监测时第 1 次采样值,k_2 表示第 k 次监测时第 2 次采样值;$CPUTotal$ 表示操作系统中用户模式、低优先级用户模式、系统模式、空闲模式、磁盘 I/O 等待、硬件中断时间之和,下标 k_1 表示第 k 次监测时第 1 次采样值,k_2 表示第 k 次监测时第 2 次采样值。

MU 定义如下:

$$MU_k=\frac{Total_k-Free_k-Buffer_k-Cache_k}{Total_k}$$

$$k=1,2,\cdots \quad (2)$$

其中,MU_k 表示第 k 次实时监测时内存利用率;$Total_k$ 表示内存总大小;$Free_k$ 表示空闲大小;$Buffer_k$ 表示 Buffer 大小;$Cache_k$ 表示 Cache 大小。

定义 3(节点能力统计指标)　定义 $Capacity$ 表示节点能力值计数,值越大,表示当前节点能力越强。

定义 4　α,β 表示强弱决定因子。

定义 5(权值矩阵调整周期)　N 表示节点调整其权值大小的周期;$Count$ 表示监测次数计数值。

3.2.2　策略描述

异构 Spark 集群下任务调度策略(HSATS)包括节点权值调整算法(NWA)与根据节点权值调度任务算法(NST)两部分。其中,NWA 算法主要是通过监测节点自身负载状况,动态地调整节点权值大小,服务于 NST 算法。NST 算法主要设计思想是优先选择权值较大节点进行任务分配,这样可以更好地发挥异构集群整体计算能力。

监测节点调整权值由获取节点信息、分析节点信息、自适应动态调整 3 个阶段组成,如图 2 所示。

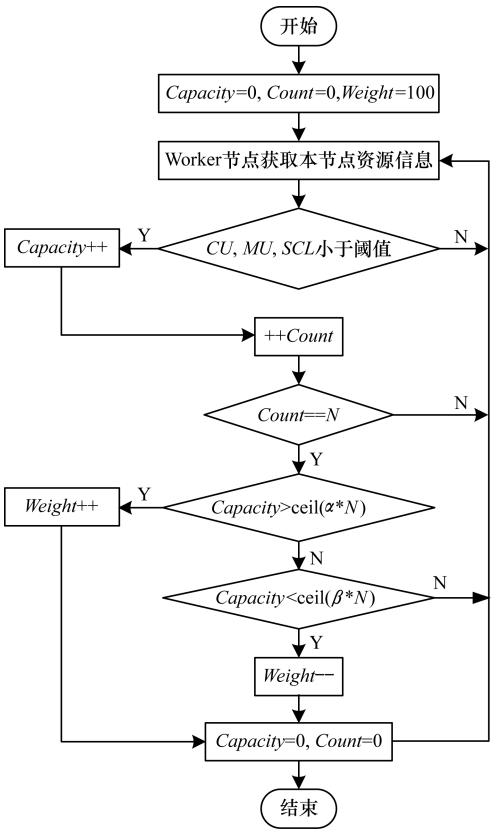

图 2　HSATS 监测调整权值流程

具体步骤如下:

(1)Worker 节点获得自身负载及资源利用情况,包括 CPU 利用率 CU、内存利用率 MU、单核平均队列长度 SCL。

(2)根据各参数的阈值,调整节点能力统计指标值大小。在此过程中,通过对比 CPU 利用率 CU、内存利用率 MU、单核平均队列长度 SCL 与各自对应阈值的大小,调整节点能力统计指标值 $Capacity$。若各参数值均小于阈值,则说明当前节点处于低负载状态,可以处理更多任务,$Capacity$ 值增加,同时,监测次数 $Count$ 值也增加;反之,则说明当前节点超负荷运行,不应再接受更多任务,$Capacity$ 值不变,只增加 $Count$ 值。

(3)判断是否达到节点权值调整周期 N。若没达到,则继续步骤(1);若达到,根据节点能力统计指标值调整节点权值大小,并重新开始循环。具体为:若 $Capacity$ 值大于强决定因子 α 与权值调整周期 N

乘积的向上取整,则说明在 N 次监测周期中,当前节点有大于$\lceil \alpha \times N \rceil$次处于低负载状态,节点能力高,可以处理更多任务,节点权值 *Weight* 增加,并将 *Capacity* 与 *Count* 置零,重新开始循环;反之,若 *Capacity* 值小于弱决定因子 β 与权值调整周期 N 乘积的向上取整,则说明在 N 次监测周期中,当前节点有小于$\lceil \beta \times N \rceil$次处于低负载状态,节点能力较低,不应接受更多任务,节点权值 *Weight* 降低,并将 *Capacity* 与 *Count* 置零,重新开始循环。

HSATS 监测节点调整权值伪代码描述如下:

算法 1 监测调整权值算法 NWA

输入 $Weight_i(i=1,2,\cdots,M)$初始化权值

输出 $Weight'_i(i=1,2,\cdots,M)$调整后权值

```
//获得节点当前负载情况
CU = getCU( )
MU = getMU( )
SCL = getSCL( )
//根据阈值调整 Capacity
if CU < C_thre and MU < M_thre and SCL < SCL_thre then
   Capacity = Capacity + 1
   Count = Count + 1
else if
   Count = Count + 1
end if
//权值调整
if Count = = N
   if Capacity > ceil(α * N)
      Weight = Weight + 1
      Capacity = 0
      Count = 0
   end if
   if Capacity < ceil(β * N)
      Weight = Weight - 1
      Capacity = 0
      Count = 0
   end if
end if
```

根据节点权值调度任务由分析节点权值、选择节点、执行调度三部分组成,具体步骤如下:

(1)任务调度节点读取各节点权值,并根据权值进行快速排序。根据节点权值排序节点后,节点能力越强,排序越靠前;反之,节点能力越低,排序越靠后。

(2)选择权值较大部分 Worker 节点,并遍历所选择出的节点,将本地化最高的 Worker 节点分配给当前 task。

本文中优先选择权值较大部分节点分配任务是因为:若集群中有 M 个节点,记每个节点计算能力为 C_i,其中 $i=0,1,\cdots,M-1$,假设对于 $\forall i>j$,满足 $C_i<C_j$,当集群在默认任务调度策略下,每个节点被选择到的概率满足均匀分布,即 $p_i=1/M, i=0,1,\cdots,M-1$,则在默认调度策略下,集群的计算能力期望为:

$$E_{\text{default}}=\sum_{i=0}^{M-1}C_i p_i \tag{3}$$

当集群在 NST 算法下,每个节点被选择到的概率为节点权值的归一化处理结果,即节点权值越大,被选择的概率越大,设此时节点 i 被选中概率为 q_i,则对于 $\forall i>j$,满足 $q_i<q_j$,则在 HSATS 算法下,集群的计算能力期望为:

$$E_{\text{NST}}=\sum_{i=0}^{M-1}C_i q_i \tag{4}$$

因此,容易得证:

$$E_{\text{default}}<E_{\text{NST}} \tag{5}$$

即在 NST 算法下,可证得系统计算能力期望整体提高。

(3)任务分配后,节点开始执行任务。

HSATS 调度任务伪代码描述如下:

算法 2 根据节点权值调度任务算法 NST

输入 $Weight'_i(i=1,2,\cdots,M)$调整后权值;任务集 *TaskSets*,大小为 T

输出 $(task_j, worker_i)$任务 j 分配在节点 i 上

```
//根据权值对节点排序
sort Weight' in descending order
for i from 1 to M'
//M' 为前 M' 个权值大的节点
    for j from 1 to T
        //获得任务在节点上的本地化参数
        get j. locality[i]
        if j. locality[i] is the maximum and flag_j = 0 then
            launch j on i
            flag_j = 1
        end if
    end for
end for
```

4 实验及结果分析

将 HSATS 与默认策略进行比较,通过在真实环境下,分析 HSATS 性能。

4.1 实验环境

实验采用 Spark-0.8.0 版本,通过将不同的任务调度策略部署在同一个集群上来进行对比。使用不同配置的 VMware 虚拟机(VM)来构建异构 Spark 集群,每台 VM 作为集群的一个节点,共 6 个节点。其中,1 台 VM 作为 Master,5 台 VM 作为 Worker,Worker 节点有 2 种类型配置。具体集群情况如表 1 所示,具体的参数值如表 2 所示。实验采用 Ganglia[12] 来监测集群状态,对不同数量集的数据进行排序来衡量 HSATS 性能。

表 1　集群相关配置

项目	配置
Spark 版本	Spark-0.8.0
操作系统	Centos 6.4
JDK	1.6
网络带宽/MB	100
Master	双核,2 GB 内存,20 GB 硬盘,数量 1
Worker Type1	双核,2 GB 内存,20 GB 硬盘,数量 3
Worker Type2	单核,1 GB 内存,20 GB 硬盘,数量 2

表 2　参数设定

参数名称	符号	默认值
检测周期/s	t	1
调整周期	N	5
强决定因子	α	0.7
弱决定因子	β	0.3
CPU 阈值	C_{thre}	0.9
内存阈值	M_{thre}	0.9
单核队列平均长度阈值	L_{thre}	5.5
节点权值初始值	$Weight_i$	100

4.2　实验结果

实验 1　本组实验的目的是验证 HSATS 与默认调度策略相比,在作业完成时间上是否有较大提升。实验排序不同规模的数据集,分别是 3 GB,3.5 GB,4 GB,4.5 GB,5 GB。为确保实验结果的可靠性,每个数据集分别进行 3 次实验,并取作业完成时间的平均值作为最终结果。实验结果如图 3 所示。

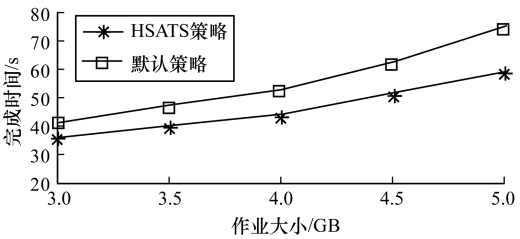

图 3　不同作业大小完成时间比较

由图 3 可以看出,针对异构 Spark 集群,在相同作业集下,HSATS 比默认的调度策略执行时间更短。并且在任务量较高情况下,HSATS 执行时间较平稳增加,而默认调度策略执行时间增加明显。由此可得,HSATS 可以更好发挥异构 Spark 集群下的计算能力。这主要是因为,在异构集群下,异构节点可以根据自身情况动态调整节点权值大小,任务调度节点在分配任务时选择任务节点更合理,从而达到自适应效果。

实验 2　本组实验的目的是观察 HSATS 与默认调度策略在进行排序过程中,随着 CPU 使用率、内存使用率及单核队列长度的不同,并分析 HSATS 是否能获得较好的性能。实验中,随机选择一个 VM 进行监测,并进行 4 GB 数据的排序操作。实验结果如图 4 所示。

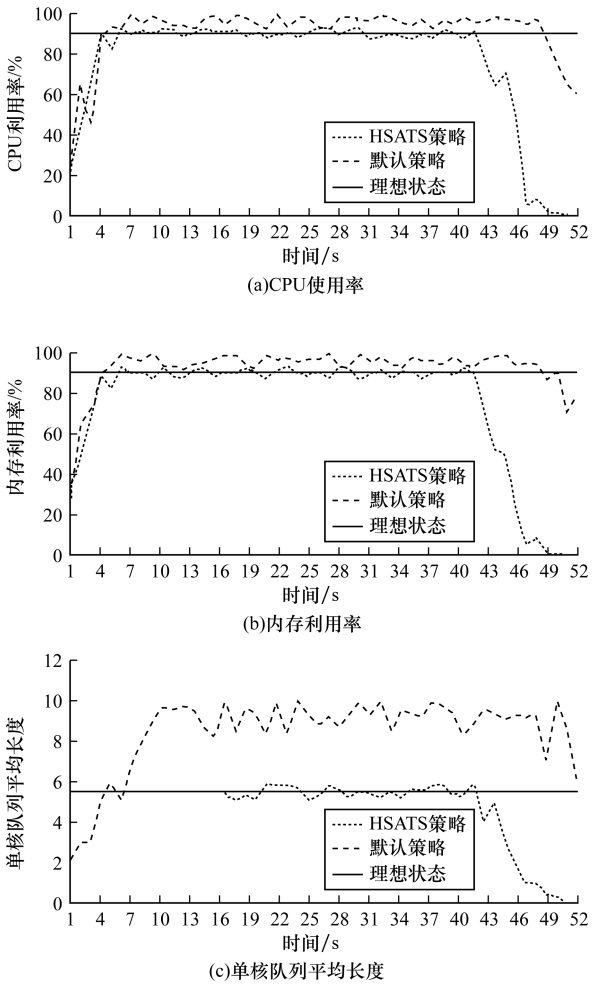

图 4　随机节点资源监测情况

由图 4 中可以看出,在默认调度策略下,系统资源利用率一直过高,CPU 利用率有时甚至可能达到 100% ,内存利用率及队列长度也长时间高于阈值。长时间的过高负载会导致系统任务执行能力的降低。而 HSATS 策略下,资源利用率一直较平稳地保持在阈值附近,这表明节点通过判断自身情况自适应调整,使自身保持在最好的工作状态下,异构节点资源利用更加合理。

综上所述,HSATS 策略针对异构 Spark 集群下的任务调度策略,能更好地提高异构集群的整体性能,提高集群资源利用率与服务质量,减少作业的完成时间。

(下转第 40 页)

5 结束语

针对传统协同过滤推荐算法的评分数据稀疏和未考虑用户兴趣的特点,本文根据用户对某一物品的偏好度,求解得到偏好值预先对评分矩阵进行填充,在一定程度上缓解了稀疏性问题。同时,考虑时间加权下的用户兴趣特点,并与其他协同过滤推荐算法进行比较,结果表明本文算法可以有效改善推荐稀疏性问题,提高推荐系统质量。下一步工作可将本文协同过滤推荐算法移植到云计算环境下,提高算法并行性和可扩展性。

参考文献

[1] 马宏伟,张光卫,李 鹏. 协同过滤推荐算法综述[J]. 小型微型计算机系统,2009,30(7):1282-1288.

[2] 孙冬婷,何 涛,张福海. 推荐系统中冷启动问题研究综述[J]. 计算机与现代化,2012,(5):59-62.

[3] Koren Y. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model[C]//Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York,USA:ACM Press,2008:426-434.

[4] 罗 辛,欧阳远新,熊 璋. 通过相似度支持度优化基于 K 近邻的协同过滤算法[J]. 计算机学报,2010,33(8):1437-1445.

[5] Luo Xin, Liu Huijun, Gou Gaopeng, et al. A Parallel Matrix Factorization Based Recommender by Alternating Stochastic Gradient Decent[J]. Engineering Applications of Artificial Intelligence,2012,25(5):1403-1412.

[6] 彭 石,周志彬,王国军. 基于评分矩阵预填充的协同过滤算法[J]. 计算机工程,2013,39(1):175-178.

[7] Wang Jun, de Vries A P, Reinders M J T. Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion[C]//Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York,USA:ACM Press,2006:501-508.

[8] 黄创光,印 鉴,汪 静,等. 不确定近邻的协同过滤推荐算法[J]. 计算机学报,2010,33(8):1369-1377.

[9] 许智宏,王宝莹. 基于项目综合相似度的协同过滤算法[J]. 计算机应用研究,2013,31(2):398-400.

[10] 邢春晓,高凤荣,战思南,等. 适应用户兴趣变化的协同过滤推荐算法[J]. 计算机研究与发展,2007,44(2):296-301.

[11] 陈志敏,李志强. 基于用户特征和项目属性的协同过滤推荐算法[J]. 计算机应用,2011,31(7):1748-1751.

[12] Baralis E, Garza P. Item Selection for Associative Classification[J]. International Journal of Intelligent Systems,2012,27(3):279-299.

[13] 韦素云,业 宁,杨旭兵. 结合项目类别和动态时间加权的协同过滤推荐算法[J]. 计算机工程,2014,40(6):206-210.

[14] Harald Steck. Item Popularity and Recommendation Accuracy[C]//Proceedings of the 5th ACM Conference on Recommender Systems. New York, USA: ACM Press,2011:125-132.

[15] 沈 键,杨煜普. 基于滚动时间窗的动态协同过滤推荐模型及算法[J]. 计算机科学,2013,40(2):206-209.

编辑 陆燕菲

(上接第 35 页)

5 结束语

本文提出基于异构 Spark 集群的任务调度策略 HSATS,该策略通过检测节点的负载及资源利用状态变化,自适应动态调整对应节点的权值,以服务于异构 Spark 集群下的任务调度。在真实环境下,通过修改源码来搭建实验平台,并进行对比实验。实验结果表明,该策略能够控制节点负载及资源利用率在较为稳定的状态,缩短了用户作业的完成时间,提高了异构集群下的资源利用率与服务质量。下一步工作是研究本文中相关参数对系统性能的影响,如节点权值的上下界问题等。同时,考虑 Spark 系统中参数调优等因素,进一步对异构 Spark 集群下任务调度策略进行优化。

参考文献

[1] The Spark Software Foundation. Spark[EB/OL]. [2015-01-08]. http://spark.apache.org.

[2] The Apache Software Foundation. Hadoop[EB/OL]. [2015-01-08]. http://hadoop.apache.org.

[3] Xu Xiaolong, Cao Lingling, Wang Xinheng. Adaptive Task Scheduling Strategy Based on Dynamic Workload Adjustment for Heterogeneous Hadoop Clusters[J]. IEEE Systems Journal,2014,(99):1-12.

[4] Nightingale E B,Chen P M,Flinn J. Speculative Execution in a Distributed File System[J]. ACM Transactions on Computer Systems,2006,24(4):361-392.

[5] Yong M, Garegrat N, Mohan S. Towards a Resource Aware Scheduler in Hadoop[C]//Proceedings of the 7th IEEE International Conference on Web Services. Los Angeles,USA:IEEE Computer Society,2009:102-109.

[6] Zaharia M, Chowdhury M, Das T, et al. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing, UCB/EECS-2011-82[R]. University of California,Berkeley,2012.

[7] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster Computing with Working Sets, UCB/EECS-2010-53[R]. University of California,Berkeley,2010.

[8] 冯 琳. 集群计算引擎 Spark 中的内存优化研究与实现[D]. 北京:清华大学,2013.

[9] 唐振坤. 基于 Spark 的机器学习平台设计与实现[D]. 厦门:厦门大学,2014.

[10] Guo Zhenhua, Fox G, Zhou Mo. Investigation of Data Locality in MapReduce[C]//Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Ottawa, Canada: IEEE Computer Society,2012:419-426.

[11] Typesafe Inc.. akka[EB/OL]. [2015-01-08]. http://akka.io/.

[12] Massie M, Li B, Nickoles B, et al. Monitoring with Ganglia[M]. Sebastopol,USA:O'Reilly Media,2012.

编辑 金胡考