CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 39 卷第 3 期 2017 年 3 月
Vol. 39, No. 3, Mar. 2017

文章编号:1007-130X(2017)03-0413-11

异构 Hadoop 集群下的负载自适应反馈调度策略*

潘佳艺[1,2,3],王　芳[1,2,3],杨静怡[1,2,3],谭支鹏[1,2,3]

(1.华中科技大学武汉光电国家实验室,湖北 武汉 430074;2.华中科技大学计算机科学与技术学院,湖北 武汉 430074;
3.华中科技大学信息存储系统教育部重点实验室,湖北 武汉 430074)

摘　要:随着基于 Hadoop 平台的大数据技术的不断发展和实践的深入,Hadoop YARN 资源调度策略在异构集群中的不适用性越发明显。一方面,节点资源无法动态分配,导致优势节点的计算资源浪费、系统性能没有充分发挥;另一方面,现有的静态资源分配策略未考虑作业在不同执行阶段的差异,易产生大量资源碎片。基于以上问题,提出了一种负载自适应调度策略。监控集群执行节点和提交作业的性能信息,利用实时监控数据建模、量化节点的综合计算能力,结合节点和作业的性能信息在调度器上启动基于相似度评估的动态资源调度方案。优化后的系统能够有效识别集群节点的执行能力差异,并根据作业任务的实时需求进行细粒度的动态资源调度,在完善 YARN 现有调度语义的同时,可作为子级资源调度方案架构在上层调度器下。在 Hadoop 2.0 上实现并测试该策略,实验结果表明,作业的自适应资源调度策略显著提高了资源利用率,集群并发度提高了 2 到 3 倍,时间性能提升了近 10%。

关键词:异构集群;监控;计算能力;动态调度;负载自适应

中图分类号:TP391　　**文献标志码**:A

doi:10.3969/j.issn.1007-130X.2017.03.002

A load-adaptive feedback scheduling strategy for heterogeneous Hadoop cluster

PAN Jia-yi[1,2,3], WANG Fang[1,2,3], YANG Jing-yi[1,2,3], TAN Zhi-peng[1,2,3]

(1. Wuhan National Lab for Optoelectronics, Huazhong Uniuersity of Science and Technology, Wuhan 430074;
2. School of Computer Science & Technology, Huazhong Uniuersity of Science and Technology, Wuhan 430074;
3. Key Laboratoryof Information Storage System, Ministry of Education,
Huazhong Uniuersity of Science and Technology, Wuhan 430074, China)

Abstract: With the development and practice of big data technology, Hadoop YARN (Yet Anouther Resource Negotiator) scheduler is no longer an effective solution in heterogeneous cluster environment. On the one hand, YARN cannot dynamically allocate the resources of nodes, which leads to a waste of better nodes' resources and poor overall system performance. On the other hand, YARN's existing static resource allocation policy ignores the difference of the different stages, which causes a large number of resource fragments. Aiming at the above problems, we put forward a load-adaptive feedback scheduling strategy. The system monitors the performance of all nodes and jobs, evaluates the computing power of each node with the real-time monitoring data. Then the scheduler starts the dynamic resource scheduling strategy based on the similarity assessment together with the monitoring information of nodes and jobs' performance. The optimized system can distinguish the heterogeneity of different nodes, allocate resources for tasks' real-time needs dynamically, refine YARN's scheduling semantics and be used as a secondary resource scheduling strategy of the upper scheduler. We implement and test

* **收稿日期**:2016-09-03;**修回日期**:2016-11-01
基金项目:**国家 863 计划**(2013AA013203)
通信地址:430074 **湖北省武汉市华中科技大学武汉光电国家实验室**
Address: Wuhan National Lab for Optoelectronics, Huazhong Univeristy of Science and Technology, Wuhan 430074, Hubei, P. R. China

the strategy on Hadoop 2.0, and the experimental results show that this scheduling strategy can significantly improve the utilization rate of resources, improve the cluster's concurrency by 2 to 3 times, and enhance the performance by nearly 10%.

Key words: heterogeneous cluster; monitor; computing power; dynamic scheduling; load-adaptive

1 引言

随着大数据与互联网时代的来临,大数据技术已成为各界研究热点。Hadoop[1]作为开源的大数据处理平台,在企业、学术领域大规模的数据处理方面均得到了广泛应用。目前,Hadoop 已成为国内外许多互联网公司的核心计算框架,如 Yahoo、Facebook 和 Twitter 等。

第 1 代 Hadoop 由计算框架 MapReduce 和底层分布式文件系统 HDFS(Hadoop Distributed File System)[2]构成,仅支持单一的 MapReduce 计算框架,其粗粒度的任务槽机制限制了资源利用率,且存在单点故障。针对 1 代 Hadoop 的问题,Apache 推出了第 2 代 Hadoop,将资源管理模块构建成为独立的通用资源管理系统 YARN(Yet Anouther Resource Negotiator)[3],统一管理集群资源。YARN 可同时支持多种计算框架(如并行计算框架 MapReduce[4,5]、流式计算框架 Storm、内存计算框架 Spark、图计算框架等)的统一资源管理,集群易于运维、数据可共享且资源可伸缩。目前 YARN 已经成为新一代资源管理的代表,腾讯 Gaia 平台、阿里云梯集群等多家互联网公司陆续构建了基于 YARN 的企业大数据平台。

资源调度器作为 YARN 最核心的组件,负责集群资源的管理和分配,所采用的资源调度策略[6]会直接影响集群的任务分配和执行效率[7,8]。相较于 1 代 Hadoop 中的粗粒度调度,YARN 提出了资源容器这一抽象概念,实现了资源隔离,一定程度上解决了第 1 代 Hadoop 并发度受限、资源利用率低的问题。但是,YARN 已有的资源调度策略仍存在以下两方面的缺陷[9]:

(1)节点异构性:实际环境中 Hadoop 多部署在异构集群上,各执行节点的计算能力存在较大差异,而 YARN 现有的资源调度语义不完善[10],为全部任务分配定额资源,在调度处理的其他环节中也忽略了节点差异,造成节点并发度基本一致,节点性能无法最大化。

(2)负载异构性:Hadoop 的使用场景[11]多为大规模数据处理,种类繁杂、特征不一,作业的实际资源需求动态变化,而 YARN 调度器为作业全部任务静态地分配资源,易出现大量的资源碎片,造成严重的资源浪费,进而影响系统资源利用率。

基于以上问题,我们提出并实现了一种针对异构 Hadoop 环境的负载自适应调度策略。本文第 2 节介绍了 YARN 现有调度策略,分析了 YARN 资源调度存在的问题;第 3 节分析了相关研究并与本文策略做对比;第 4 节归纳了负载自适应反馈调度策略的总体设计,并对各个模块的方案设计进行了详细阐述;第 5 节利用多种 Hadoop Benchmark 测试验证了该作业自适应资源调度策略的效果;最后对全文进行了总结。

2 异构环境下的 YARN 调度

YARN 系统包括资源管理器 RM(Resource Manager)、作业控制器 AM(Application Master)和节点管理器 NM(Node Manager)三方。RM 负责集群资源管理,AM 负责管理作业执行,NM 负责执行节点的资源管理。YARN 的资源调度基于这样的假设:

(1)执行节点的能力相同,以相同速度执行作业。

(2)同类任务具有相近的工作时间和资源需求。

基于这样的同质假设,YARN 现有的调度语义是静态的。如图 1 所示,YARN 资源调度器会根据上层作业任务提出的资源需求进行调度[12],提供一个抽象的资源封装:资源容器。不同于 Hadoop 1.0 中粗粒度的任务槽,资源容器实现了资源隔离(目前已实现内存、CPU 的隔离)。任务的资源容器请求规格参数配置在集群配置文件中,客户端提交作业后,作业控制器读取配置文件,此后在该作业运行过程中,其任务资源请求规格不可再修改。MapReduce 作业提交后,会被划分为多组 map 任务和 reduce 任务,在作业运行过程中,运行在不同节点上的同类任务(map 任务或 reduce 任务)申请的资源规格始终是固定不变的。此外,YARN 原生的容量调度器和公平调度器支持多队列调度,在多队列调度配置下同时提交的多项作

业,其任务的资源请求额度也是固定且相同的。

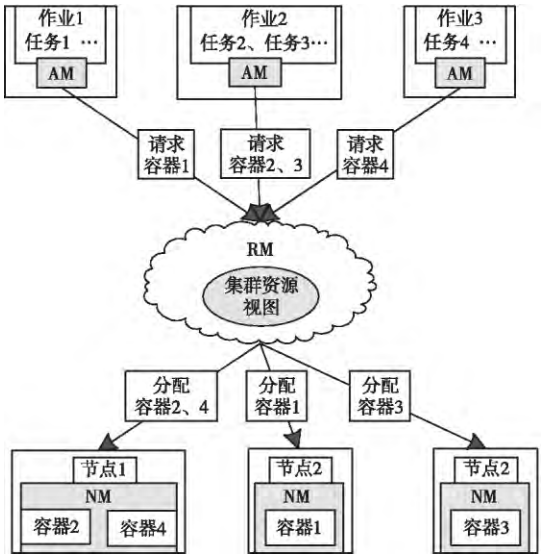


Figure 1 YARN resource management

图 1 YARN 资源调度图

但是,这样的同质假设在现实世界的运行环境中往往是不成立的,这导致 YARN 的资源管理在异构环境中存在诸多局限性。

实际运行环境中,Hadoop 集群多为异构集群,通常拥有数千至上万个配置各异的节点,各执行节点的综合性能(计算能力)[13]存在较大差异,而 YARN 现有资源调度器的调度语义不完善,始终按照默认规格为作业调度资源,在不同节点上按相同规格为同类任务划分资源。这种静态调度无法根据异构集群中节点的计算能力、作业实时执行状况和资源实际使用情况动态调整节点上的资源容器划分和承担的任务份额,间接导致集群中优势与劣势执行节点上的最大并发度一样。例如,YARN 集群默认配置下每台机器为集群提供 8 GB 内存,而默认情况下,每个任务需申请 1 GB 内存才能投入执行,因此每个节点上最多能同时运行的任务数为 8,其数值只会随着任务执行的成功、失败等因素变化,无法根据集群节点实时计算能力的高低动态调整。优势节点的性能无法最大化,资源未被充分利用;而劣势节点上容易负载过重,造成节点性能下降。从而导致系统资源利用率下降,作业整体完成时间延长。

Hadoop 的使用场景一般是大规模数据处理,这种类型应用通常种类繁杂、特征不一,同类作业的任务实际资源需求随时间动态变化,不同类作业之间的资源需求也存在差异。而 YARN 现有资源调度策略始终为任务静态地分配统一规格的资源,未考虑不同执行阶段作业任务需求的差异性和不同类作业间的差异,为了使作业能够运行通过,集群默认的资源请求规格往往超过了任务的实际资源需求。这种资源请求与实际需求之间的差异,容易引发系统出现大量的资源碎片,进一步使系统整体资源利用率下降。例如,有些作业的任务(如机器学习、数据挖掘作业等)可能是 I/O 密集型[14]的,消耗的 CPU 资源非常少,如果此时资源调度器为该任务分配一整个单位的 CPU,而其实际的 CPU 利用率很低[15],这也是一种严重的资源浪费。

为了验证在 YARN 现有的静态资源调度机制下,集群可能存在的大量资源碎片问题,本文基于 Hadoop 集群进行了词频统计程序 wordcount 等多种 benchmark 样例的测试,对作业的整个运行过程进行全程监控,统计了资源容器在作业执行过程中内存资源和 CPU 资源的实际使用情况。

实验采集了词频统计程序 wordcount 在 59 个监控周期内的资源消耗情况,如图 2 所示,单个资源容器在作业执行过程中的实际内存资源消耗存在一定波动,但总体比较稳定,均值在 240 MB 左右。而 YARN 默认的资源容器内存使用量为 1 024 MB。资源容器被分配后,NM 认为节点上分配出了 1 024 MB 的内存,即使实际任务只消耗了 240 MB 左右,剩余内存也不能被其他任务使用,只能保持空闲,直到所属的容器结束运行,被 NM 释放。相较于内存消耗,CPU 的资源消耗浮

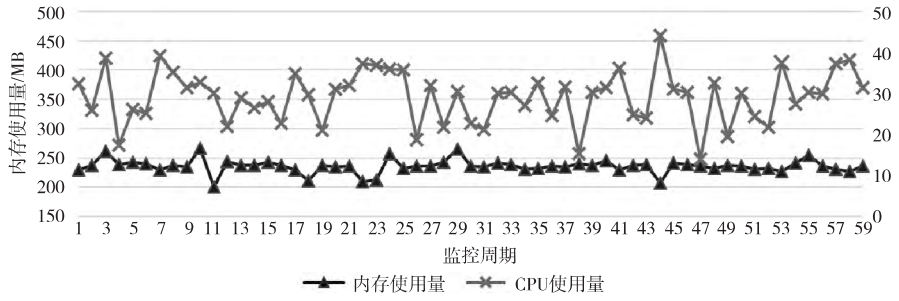


Figure 2 Container resource consumption of wordcount

图 2 wordcount 中容器的资源消耗

动较大,但跟系统默认分配的一核 CPU 相比仍有明显差距,也存在一定的资源碎片。

在大数据环境下,Hadoop 集群中有大量的任务并发执行,每个任务都与一个资源容器对应,在这种统一规格的静态资源分配机制下,几乎每个任务都产生资源碎片,整个系统存在大量的未利用资源,从而导致集群并发度大幅降低、资源利用率下降。

此外,不同类型的作业运行过程中,任务消耗的内存、CPU 资源量也不同。修改 YARN 的资源请求配置,压缩内存请求规格后进行样例测试发现,内存请求压缩到一定程度后,任务的实际资源请求无法得到满足,造成大量的任务失败后,作业运行失败。多次压缩测试可以探测到作业成功运行所需的最低内存额度。针对不同的样例程序,探测到的资源需求下限也不相同。例如 wordcount 程序在内存压缩到 320 MB 时已经开始出现失败;而对于 TestDFSIO 程序,即使将内存压缩到 256 MB,依然可以成功运行。这说明不同类型、不同规模的作业对资源的需求也是不同的,为了精准地预测任务的资源需求,需要结合上层作业监控。如何根据作业任务的实际运行情况确定后续的资源分配是本策略的研究重点。

3 相关工作

针对 Hadoop 的资源分配机制在异构环境中的局限性问题,众多企业、许多高校和科研机构近年来进行了相关研究。

文献[16]介绍了一种基于性能驱动的任务调度算法,基于任务的完成时间、剩余任务数目等信息评判作业的优先级,并根据作业优先级队列动态地进行任务调度和资源分配。该算法考虑了作业执行情况但忽略了集群中节点的异构性。

与本文策略类似,文献[17]基于 CPU、I/O 利用率对 MapReduce 作业负载进行了分类。该算法专注于作业调度层,利用多作业队列为不同类负载划分不同规格的资源,一定程度上补充了原 Hadoop 在负载异构性上的缺陷,但忽略了对节点能力的考量。

Zaharia 等人[18]提出了 LATE(Longest Approximate Time to End)调度器,主要针对 Hadoop 1.0 的同质假设所造成的异构环境下的错误预测执行进行优化,优先启动最快完成的预测任务。LATE 考虑到节点的异构性,用节点上运行任务进度总和识别节点的快慢,只在快节点上启动预测任务。相较于 Hadoop 自带调度器,该调度器在异构环境下更健壮。但是,LATE 算法中静态地估算任务完成时间,结果不够精确,也没有根据任务类型的不同分类划分快慢节点。文献[19]对 LATE 调度算法进行了优化,提出了 SAMR(Self-Adaptive MapReduce)调度。利用节点上的任务执行历史信息去调节 map、reduce 任务每个子阶段的时间比重,动态地计算任务进度和执行速率。针对 map 任务和 reduce 任务分类划分集群快、慢节点。LATE 调度和 SAMR 调度都关注了 Hadoop 的节点异构性问题,在 Hadoop 1.0 上实现了优化。但是,在任务调度层面,只关注了预测任务的执行,对于普通任务在异构节点上的调度没有涉及。

Throughput 调度器[20]利用节点容量信息和贝叶斯主动学习到的作业需求之间的匹配,在节点上动态调度,以缩减异构集群上作业的完成时间。但是,该方法假设节点能力不会动态变化,采用离线执行探测作业的方法获取节点能力信息。

如表 1 所示,相较于本文的策略,这些研究有些考虑到了负载的异构性,对作业整体的负载类型作区分,但没有考虑作业内部不同任务之间的异构性。有些比较全面地考虑了负载异构性,但对节点计算能力的量化是离线的或者静态的,结果不准确。此外,大多数研究没有考虑到节点异构性带来的碎片化问题。而我们提出的负载自适应调度策略,综合

Table 1 Comparison of related research characteristics

表 1 相关研究特征对比表

特征	作业间负载分类	作业内负载分类	量化节点计算能力	资源规格调整	所在调度层
文献[16]	有	无	无	无	任务调度层
文献[17]	有	无	无	有	作业调度层
LATE	无	无	静态	无	任务调度层
SAMR	无	无	动态	无	任务调度层
Throughput	有	有	离线	无	作业调度层
本文策略	有	有	动态	有	任务调度层

集群负载特性和任务进度等信息,实时地对节点计算能力进行量化,全面考虑了节点异构性问题。根据任务类型和负载类型动态地调整资源分配的规格,解决了负载异构性带来的资源碎片化问题。

4 负载自适应反馈调度原理

如图 3 所示,作业的自适应资源调度策略的整体架构由客户端、Hadoop 集群以及基于 Ganglia 的监控服务器三方组成。集群启动后 NM 和 AM 分别启动后台监控,收集集群所有执行节点的节点性能监控信息、作业特性监控信息,汇总分析后形成原始监控文件,实时发送给监控服务器。用户在客户端向 Hadoop 集群提交作业后,监控服务器上的计算能力评估模块根据原始监控文件中的作业环境信息选取相应的计算能力评估模型进行建模,结合节点性能和作业运行情况综合地评估分析当前时刻节点计算能力的优劣。监控服务器定期将处理后的监控信息和节点计算能力评估结果整合为监控文件反馈给集群主节点。根据监控文件中包含的作业历史资源用量、节点实时计算能力和节点实时性能表现,调度器启动基于相似度评估的动态资源调度方案。可分为以下四个组件:

(1)节点性能监控:部署在集群所有执行节点上,用于对集群执行节点进行全方位的监控,旨在通过详细、实时的软、硬件监控数据为后续评估节点计算能力提供依据。

(2)作业特性监控:部署在集群所有执行节点上,采集作业执行的相关多项监控信息,为后续资源反馈调度提供依据。

(3)计算能力评估:部署在集群监控服务器节点上,利用采集的多项监控数据评估执行节点的计算能力,量化集群所有节点计算能力的相对优劣度。

(4)资源反馈调度:部署在集群主节点调度器内,结合各项监控信息,在优势节点上启动基于相似度评估的资源调度策略,通过调节资源容器的大小,控制任务从节点上拉取到的资源份额。

该策略在原有的 Hadoop 平台上,加入了全面的监控技术、计算能力评估技术以及动态资源反馈调度技术。相较于原 Hadoop 系统,优化后的系统具有全面的节点监控、细分的作业监控,为资源调度的调整提供了可靠的数据支持;增加了资源反馈调度后的资源分配更加合理,最大程度上避免了资源碎片化、缩短了系统响应时间、提高了 Hadoop 平台整体执行效率;完善了 YRAN 现有的调度语义,支持更细粒度的调度,促进任务的高效并发;同时具有良好的移植性,可作为子级资源调度方案部署在 YARN 多种自带资源调度器下。

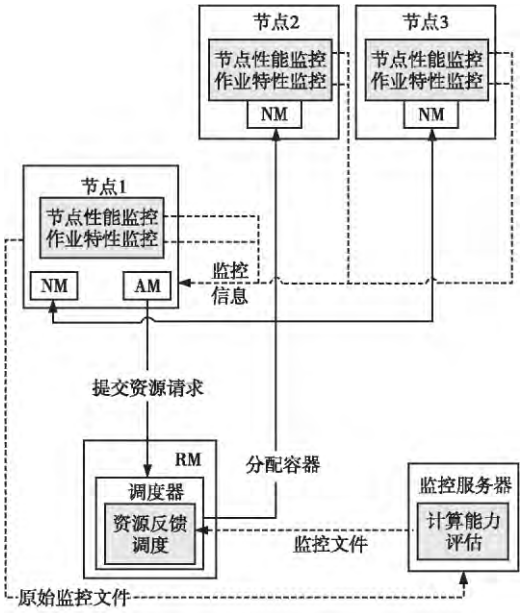


Figure 3 Architecture of load-adaptive feedback scheduling

图 3 负载自适应反馈调度架构图

4.1 节点性能监控

异构集群中,节点性能与节点硬件配置有着密切关联,我们选取最通用的四项指标:CPU 空闲率 *cpuFRate*(*cpuFreeRate*)、内存空闲率 *memFRate*(*memFreeRate*)、网络空闲率 *netFRate*(*netFreeRate*)和磁盘空闲率 *diskFRate*(*diskFreeRate*)作为节点的硬件性能监控内容。执行节点上硬件资源的空闲率越高,说明有越多的可用资源,有能力高效地运行更多的任务而不会造成负载间的资源争用。因而倾向于认为资源空闲率高的节点计算能力更高,节点性能更好。

此外,节点性能也与在 Hadoop 上的任务执行状况密切相关。参照 LATE 算法,本文提取了三项软件指标进行动态的节点监控,以反映任务的执行效率。

(1)*avg_wTime*(*avg_waitTime*)等待该节点上出现下一个空闲容器的平均等待时间。空闲容器的出现代表着一个新的任务即将投入运行,因此该指标直接反映了集群各执行节点能够投入后续任务的快慢情况,体现了节点执行效率的差异。等待时间越短的节点上任务运行得更快,应该分配到更多的任务。

(2)*tFRate*(*taskFailureRate*):节点上任务执行失败率。该项指标表示执行失败(包括 child

jvm 失败、child task 失败、任务超时、节点崩溃等)的任务占监控周期中全部启动任务的比例。该比例越低,节点越可能成为优势节点,集群将趋向于在该节点分配更多的任务以增加节点的并发度。

(3)avg_rTime($avg\_responseTime$)节点上任务执行的平均响应时间。该项指标表示节点上从任务启动到完成的耗时均值,一定程度上反映了节点在异构机器环境下的计算能力大小。响应时间越短,节点的计算能力越强。

节点性能的监控由 Ganglia 监控平台和各执行节点共同完成。自定义的节点软件指标则利用 gmetric 规范采集:AM 端部署的后台监控线程在每个监控周期内解析执行节点上的任务实例、采集该节点的软件指标,最终汇总成包含全部执行节点软件信息的多级映射表。节点监控信息最终全部汇总在 AM 本地,监控服务器周期性地从 AM 上读取节点监控数据并实时更新到 RRD 数据库以及本地监控文件中,为后续计算能力评估以及资源反馈调度提供数据依据。

4.2 作业特性监控

该项监控主要包括提交作业后预判作业环境、监控作业任务分段时长、监控作业资源消耗信息。

预判作业环境:不同类型作业对系统资源需求的份额也有所不同。而在不同的作业环境下节点表现出的计算能力也有所不同。计算资源充足的节点在 CPU 密集型环境下可能表现出优于其他节点的良好性能,但在 I/O 密集型的环境下有可能表现得差强人意。因此,对集群不同时刻的作业环境加以区分,对准确评估节点的实时计算能力有着重要意义。为此,我们在作业运行过程中从底层角度对上层作业进行分类,通过对节点读、写特性的数据收集,分析、预判当前作业环境,进而选择合适的评估模型,保证资源需求得到满足。一般认为在一个周期内,集群节点的 I/O 操作平均时间超过了整体操作时间的 50%,就可判定当前集群作业环境为 I/O 密集型,否则集群作业环境为 CPU 密集型。

监控作业任务分段时长:Hadoop 2.0 中,MapReduce 作业任务划分了多个子阶段。Map Task 可分为 map、sort,Reduce Task 可分为 copy、sort 和 reduce。不同的子阶段在任务中占据的比重不同,为了得到更为精确的任务响应时间,我们定期统计运行中的 Map Task 和 Reduce Task 中已完成子阶段的响应时间,按各子阶段的权重综合计算任务的响应时间,据此衡量作业在当前节点上的稳定性。

监控作业资源消耗信息:YARN 中每个任务都运行在一个容器中,由 NM 对节点上的容器进行资源管理和监控。集群运行中,NM 上启动容器监控器,定期监控每个资源容器的实际资源(内存、CPU)消耗信息。在 Hadoop 中,资源容器的份额取决于用户的配置,而容器监控器监控到的资源消耗才是该任务实际所需要的资源份额。监控信息头部含有该容器对应的任务标识、所属作业标识和节点标识,用于后续反馈调度时的匹配。MapReduce 作业同类任务由于完成的功能相近,其资源消耗有相似性。可以用某类任务的资源消耗历史信息来预测下一周期内该类任务的实际资源需求。分类监控 map、reduce 任务的资源消耗信息,为后续任务配置合理的资源份额,避免资源浪费和大量资源碎片的产生。

4.3 计算能力评估

当前作业环境、节点硬件处理能力和实时软件执行效率都对执行节点的实时计算能力有着不可忽视的影响。结合以上信息,我们设计了一个通用计算能力评估模型[21]。

$$avg\_rTime = F_j(memFRate, cpuFRate, netFRate, diskFRate, avg\_wTime, tFrate) \quad (1)$$

选择平均响应时间 avg_rTime 来反映节点在当前环境下的计算能力。通过在离线环境下多次运行特征作业,对收集到的历史数据进行多元线性回归,最终选择拟合度较高的模型作为最终的计算能力评估模型。该模型的参数显示了硬件资源空闲率、平均等待时间等指标对节点计算能力的影响权重。其中软件监控信息可从 Ganglia RRD 数据库中直接获取,硬件监控信息则为 RRD 数据库中读取的资源空闲和总量作商归一化后得到的实时计算值。

计算能力评估的目的是判断当前 Hadoop 集群作业环境下的性能优势节点和劣势节点,而在不同作业环境下各项指标对节点计算能力的影响程度大不相同。因此,为了尽可能精确地评估节点计算能力,需要根据上层作业类型的不同,建立不同的评估模型。目前主要的作业环境大致分为两类:CPU 密集型和 I/O 密集型。为了提高评估结果的精确性,在离线环境下分别执行 CPU 密集型作业和 I/O 密集型作业,得到针对不同负载环境的两种评估模型。

作业运行过程中,监控服务器定期收到整合后的监控文件,其中包含作业环境的预判和评估模型

所需的向量信息。评估模块可据此选择相应的模型进行集群节点计算能力的估算,最终得到集群节点计算能力评分均值 $avg\_cluTime$。集群中大部分节点如式(2),针对节点 N_j 量化出的实时计算能力 avg_rTime_j,根据 avg_rTime_j 与 $avg\_cluTime$ 的数值关系判定节点 N_j 的性能优劣,并将结果写入本地监控文件。其中 β 取经验值 60%。

$$N_j=\begin{cases}\text{优势节点}, avg\_rTime_j \geqslant \\ \quad \beta * avg\_avg\_cluTime \\ \text{劣势节点}, avg\_rTime_j < \\ \quad \beta * avg\_avg\_cluTime\end{cases} \tag{2}$$

后续的反馈调度中,节点优劣性是是否在该节点上启动反馈调整的决定性因素。计算能力评估结果的精确性影响着后续反馈调整的有效性。该模块中节点计算能力的评估是动态的,每个新的监控周期,监控服务器都会根据收到的最新监控文件进行一轮新的节点计算能力估算和优劣节点判别。硬件故障或者作业环境的改变对节点能力造成的影响都可以在合理的监控间隔内被捕捉到,进而保证了后续调度的有效性。

4.4 资源反馈调度

YARN 采用双层调度模型,支持多级队列的调度器(如容量调度器、公平调度器)按队列调度策略选择待调度的作业。本文仅关注下层的任务调度,忽略调度器对待调度作业的选择。资源反馈调度的核心在于资源容器规格的反馈调整。容器的请求和分配与 RM、AM、NM 三方息息相关,因此选择合适的时刻进行反馈调整,直接影响策略的性能表现。

作业提交后,AM 开始以五元组〈$priority$, $host$, $capability$, $containers$, $relax\_locality$〉的形式通过周期心跳向 RM 批量提交任务实例[22]的资源请求,提交后的资源请求被 RM 管理,根据资源分配的实际情况不断修改。

以 MapReduce 作业[23]为例,提交后 AM 为之创建多组 Map Task 和 Reduce Task。当一个 Map Task 请求节点 N_i 上的资源容器同时节点 N_i 利用周期性心跳向 RM 上报了 NODE_UPDATE 事件后,RM 尝试在 N_i 上分配该任务,但如果 N_i 剩余资源不满足该任务的资源需求,资源调度器中设计了预留机制[24](避免了因节点资源不足造成的饿死现象[25]),任务会等待直到 N_i 上出现足够的资源。因此,资源请求的提交和对应的资源分配之间往往存在大量时延。以 10 GB 规模的 wordcount 为例,任务请求的提交和任务开始运行之间最多可间隔几百秒。

由于本策略的监控信息都是实时的、针对特定监控周期的,如果在 AM 发送资源请求之前进行资源容器规格的反馈调整,所采用的节点计算能力信息和实际的资源分配时刻之间相差数个监控周期,而在这期间作业环境和节点的性能信息极有可能发生较大的变化,造成对节点能力的错误预估,最终导致节点上资源分配不合理,无法最大化地发挥本策略的优化效果。为了性能的最大化,应选择与实际资源分配最接近的时刻进行资源反馈调节。因此,我们将该调度插入在 RM 端子队列调度器下,在子队列创建新的资源容器之前对该容器的规格进行反馈调节。利用历史周期内作业的资源消耗情况预测后续任务的实际资源需求。

劣势节点的计算能力相比于集群整体计算能力有明显差距,一般有较长的等待时间和较高的任务失败率。在劣势节点上调整资源容器的规格、提供任务并发度,极有可能出现更多的失败任务,进而引发节点性能的进一步下降。为了避免这种现象,我们根据当前节点的计算能力在线评估结果决定是否在本次分配中开启反馈调整。仅当该节点在当前周期被判定为优势节点时,才启动反馈调整。

此外,为了在保证作业执行稳定、节点性能无过度波动,通过计算同一节点在相邻监控周期中两个特征向量的相似度来评估节点的性能变化。该反馈调度可以容忍一定程度的节点性能下降,并尝试采用回退的方法改善节点性能。但是,当节点性能的下降超过一定阈值后不再容忍,暂时停止反馈调整,保持 YARN 默认的资源配置策略,避免过度的调整造成节点性能的进一步下降。具体算法描述如下。

算法 1 资源反馈调度算法

输入:监控文件;

输出:新分配的资源容器。

Step 1 节点 N_i 向 RM 发送周期性心跳,并触发 NODE_UPDATE 事件。

Step 2 RM 中调度器检查节点 N_i 资源合理性,若 N_i 有可用资源,转 Step 3;否则,本次心跳结束。

Step 3 按自带调度器原始策略为节点 N_i 选择子队列和待分配作业,尝试为该作业分配资源,创建新的容器(设当前被选定的作业为作业 j,对应的作业序列号为 job_j)。

Step 4 反馈调度器按 job_j 读取本地监控文件,检查文件中监控周期 ID 的合理性。若为非法 ID,转 Step 10。

Step 5 当前监控周期为 k,按节点将文件信息以多元向

量形式逐条汇总到调度器中。以节点 N_i 为例,向量形式如下。

$$\boldsymbol{r}_{jik}=(x_1^k,x_2^k,\cdots,x_9^k) \tag{3}$$

其中,$x_1^k\sim x_5^k$ 依次表示在监控周期 k 下,作业 j 在 N_i 上执行 Map Task 的子阶段 map、sort 和 Reduce Task 的子阶段 copy、sort、reduce 的平均完成时长,$x_6^k\sim x_9^k$ 分别表示该周期下作业 j 在 N_i 上执行 Map Task 和 ReduceTask 的平均内存和 CPU 消耗。

Step 6 检查节点 N_i 在当前周期内的优劣性,若 N_i 为劣势节点,转 Step 10;否则,从调度器中提取前一非空监控周期 m 中节点 N_i 的信息。

Step 7 利用余弦公式计算两个监控周期下作业任务分段时长监控信息的相似度,判定作业 i 在当前执行节点上的性能是否稳定。

$$\Omega=simcos(\boldsymbol{r}_{jik},\boldsymbol{r}_{jim})=\frac{\sum_{j=1}^{5}(x_j*x'_j)}{\sum_{j=1}^{5}x_j^2*\sum_{j=1}^{5}x_j'^2} \tag{4}$$

Step 8 若 $\Omega\in[\varphi,1]$(φ 取经验值 0.5),认为作业执行稳定,可继续调整,选定周期 m 作为当前监控周期 k 的基准参考周期;若 $\Omega\notin[\varphi,1]$,按监控周期回溯遍历调度器中的所有记录,寻找与当前周期信息相似度最大且满足要求的周期 n,作为基准周期。如果基准参考周期不存在,转 Step 10。

Step 9 根据基准周期下作业 j 在节点 N_i 上资源使用情况,对 Map 型资源容器和 Reduce 型资源容器,资源配置分别为:

$$map\_container=(\varepsilon*x_6^k,\gamma,x_7^k) \tag{5}$$

$$reduce\_container=(\mu*x_8^k,\sigma,x_9^k) \tag{6}$$

转 Step 11。ε、γ、μ、σ 为弹性调整因子,可依据用户经验值自行设定。

Step 10 对 Map 型、Reduce 型的资源容器,资源份额统一配置为系统默认规格(〈1 024 MB,1 core〉)。

Step 11 为作业 j 分配新的资源容器并通知 NM、AM 端,心跳结束。

5 测试分析

该负载自适应反馈调度策略针对 Hadoop YARN 资源调度中存在的以下两个问题所设计:

(1)调度器无法根据集群节点综合计算能力的优劣动态调整任务份额;

(2)资源容器规格固定,存在资源碎片,资源利用率低。

为验证负载自适应反馈调度策略在 Hadoop 上的整体优化性能,本节针对大数据环境下 CPU 密集型、I/O 密集型的典型作业进行了性能测试。从调度器能否根据节点性能动态调整任务份额、能否根据任务实际需求分配资源两方面进行方案验证。

5.1 测试环境

测试平台由 Hadoop 2.0 和 Ganglia 组成,Hadoop 异构集群由一台 Master 节点和三台 Slave 节点构成,各终端通过 40 Gbps 端口速率的 IB 交换机互联。主要采用 Bigdata miroBench 中 wordcount benchmark、sort benchmark、terasort benchmar 三种测试工具进行性能测试[26,27]。为避免频繁监控带来不必要的性能损失,选取 30 s 作为监控周期,弹性调整因子选为 1.2。机器基本的软硬件配置如表 2 和表 3 所示。为保证测试的精确度和准确性,以下实验数据均为同一组参数下 10 次重复测试所得平均值。

Table 2 Basic machine software configuration

表 2 机器基本软件配置表

配置项	配置版本
操作系统	RHEL5.6
网络配置	1.7.0_79
JDK 版本	Hadoop-2.7.1
Hadoop 版本	Ganglia-gmond-3.1.0
Ganglia 版本	Mellanox InfiniBand QDR 40Gb/s NIC

Table 3 Basic machine hardware configuration

表 3 机器基本硬件配置表

配置项	内存/GB	处理器	硬盘	可用磁盘空间/GB
主节点	23	2*Intel(R) Xeon(R) E5620	SAS Disks (15 000 RPM,300 GB)	162
执行节点 1	20	2*Intel(R) Xeon(R) E5620	SAS Disks (15 000 RPM,300 GB)	162
执行节点 2	18	2*Intel(R) Xeon(R) E5620	SAS Disks (15 000 RPM,300 GB)	196
执行节点 3	23	2*Intel(R) Xeon(R) E5620	2*SAS Disks (15 000 RPM,300 GB)	522

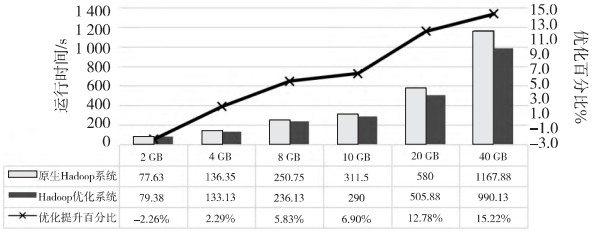
5.2 作业运行时间

5.2.1 CPU 密集型作业测试

选用词频统计作业 wordcount 做 CPU 密集型测试。在测试系统中,分别对原 Hadoop 系统和实现了本策略的 Hadoop 优化系统进行不同数据规模的 wordcount 基准测试,数据规模分为 2 GB、4 GB、8 GB、10 GB、20 GB 以及 40 GB。

图 4 显示:在 2GB 左右的数据规模下,wordcount 作业运行时间通常不足 80 s。而为了收集历史数据,反馈调度在一个监控周期之后才开始生效。因此,在 Hadoop 优化系统上的运行时间反而比原 Hadoop 系统长。在 4 GB 以上的数据规模,Hadoop 优化系统的时间性能优于原 Hadoop 系

统;随着处理数据规模的增大,Hadoop 优化系统上的时间性能优化效果越来越明显。这是因为对于较大的数据规模,作业的处理时间更长,监控周期更多,监控系统对作业的预测也更准确;同时数据规模增大导致并发任务数增加,使得本策略的调度效果更加明显。相比于原系统,本策略的动态调度极大程度上减少了系统资源碎片,提升了资源利用率的同时增加了集群节点并发度,有利于集群上作业的快速响应、Hadoop 整体执行效率的提高。



	2 GB	4 GB	8 GB	10 GB	20 GB	40 GB
原生Hadoop系统	77.63	136.35	250.75	311.5	580	1167.88
Hadoop优化系统	79.38	133.13	236.13	290	505.88	990.13
优化提升百分比	−2.26%	2.29%	5.83%	6.90%	12.78%	15.22%

Figure 4 Time performance optimization of wordcount

图 4 wordcount 时间性能优化图

5.2.2 I/O 密集型作业测试

sort 排序:排序的原始数据由 Randomwriter 作业产生,在该测试系统中,三个执行节点共产生 30 GB 数据,sort 程序将其划分成 480 个 Map 任务进行排序。多次重复测试所得结果如图 5 所示,Hadoop 优化系统对 sort 排序的优化效果比较稳定,执行时间性能的优化稳定在 14%附近,较为显著地提升了系统资源的利用率。

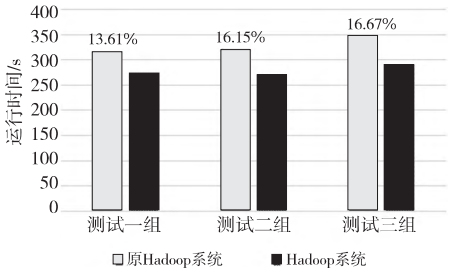
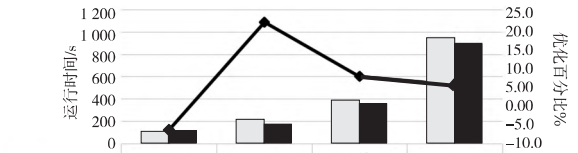


Figure 5 Time performance optimization of sort

图 5 sort 时间性能优化图

terasort 排序:排序的原始数据由 teragen 作业产生,分别选取 2 GB、4 GB、8 GB 和 16 GB 大小的数据集测试 terasort 排序。为保证实验精度,重复多次测试取平均值,测试结果如图 6 所示。对 2 GB 规模的数据,由于规模小、任务少,程序运行时间大多消耗在 Reduce 阶段,Hadoop 优化系统的运行时间反而更长,程序未得到优化。对 4 GB 以上的数据规模,Hadoop 优化系统显示出了一定程度的优化。但是,terasort 受机器网络、磁盘等因素影响大,集群执行节点性能不稳定性,优化效果不稳定,幅度在 5%～22%。



	2 GB	4 GB	8 GB	16 GB
原生Hadoop系统	112	223	389.33	956
Hadoop优化系统	119.2	174.33	359.67	905.5
优化百分比	−6.42%	21.83%	7.62%	5.28%

Figure 6 Time performance optimization of teraesort

图 6 tearsort 时间性能优化图

5.3 节点并发度

如表 4 所示,在 20 GB 大小的 wordcount 作业执行过程中,优化后 Hadoop 系统的任务并发度在 8～20 附近,相较于原 Hadoop 系统有了大幅度提升,监控稳定后任务并发度最多可提升至原来的 2～3 倍。原 Hadoop 系统中,集群执行节点上最大的任务并发度为 8 且不可调节(YARN 默认配置每个节点占用机器 8 GB 内存,每个任务默认申请 1 GB 的内存,因此任务最大并行量为 8,该值会随着任务执行状态变化,但调度器无法根据节点能力差异控制任务分配的份额),而优化后的系统在完善了现有调度语义的同时增加了集群各执行节点的任务并发度。优化后的系统能够根据节点当前综合性能的优劣以及作业的执行情况动态调整并发任务数,让优势节点承担更多任务,同时避免将过多任务投放在性能较差的节点,从而导致劣势节点原有负载加重、开启更多的备份任务,进一步引发集群作业响应的延迟。

5.4 资源利用率

原 Hadoop 系统中,集群作业每一个任务申请到的内存资源规格都是 1 024 MB 并且静态不可调,而任务实际消耗资源量却远少于此,作业任务占有的部分内存资源得不到利用,系统资源利用率因此降低、资源碎片现象严重,进一步导致集群节点并发度下降。

而在实现了作业的自适应资源调度策略的 Hadoop 优化系统上,调度器能够根据作业任务的实际资源需求动态地进行资源分配。如表 5 所示,在 20 GB 大小的 wordcount 测试中,优化后的系统将内存分配额度压缩到了 300 MB 附近。相较于原 Hadoop 系统,优化后的 Hadoop 系统在大多数监控周期内将内存分配需求压缩到原来的 1/3 左右,提升了资源利用率的同时增加了集群节点并发度,有利于集群上作业的快速响应、Hadoop 整体执行效率的提高。该 wordcount 程序运行在原系统上需要运行16个周期,而运行在Hadoop优

Table 4 Comparison of node concurrency in part of the cycle

表 4 wordcount 部分周期节点并发度对比表

监控周期	原 Hadoop 系统			Hadoop 优化系统			并发度提升比/%
	节点 1	节点 2	节点 3	节点 1	节点 2	节点 3	
1	8	8	8	8	8	8	0
4	6	8	7	19	10	19	228.6
8	8	8	8	21	15	21	238.5
12	6	8	8	4	3	3	-
16	1	1	2	已结束	已结束	已结束	-

Table 5 Comparison of memory assignment in the part of cycle

表 5 wordcount 部分周期节点内存分配对比表

监控周期	原 Hadoop 系统			Hadoop 优化系统			内存分配压缩比/%
	节点 1	节点 2	节点 3	节点 1	节点 2	节点 3	
1	1 024	1 024	1 024	310.84	258.6	308.17	28.56%
4	1 024	1 024	1 024	307.52	303.88	308.35	28.94%
8	1 024	1 024	1 024	272.75	309.73	282.50	28.16%
12	1 024	1 024	1 024	307.01	311.00	310.8	30.23%
16	1 024	1 024	1 024	已结束	已结束	已结束	-

化系统上只需要 12 个周期就可以完成。由于合理的资源使用、任务分配,优化后的系统提前于原 Hadoop 系统 4 个监控周期完成。

6 结束语

随着大数据技术的发展,Hadoop 的应用越来越广泛、实践越来越深入,Hadoop YARN 资源调度器在异构集群中的局限性也越发明显。一方面,YARN 现有资源调度器无法根据异构集群中节点计算能力的差异、作业执行情况的变动和资源的实际使用情况动态调整资源分配的份额,静态分配使得优势节点性能没有得到充分发挥,而劣势节点上的负载过重,负载的分配不合理;另一方面,大数据应用种类繁杂、特征不一,不同种类的作业资源需求不同,即使同一作业在运行过程中资源需求也在动态变化,YARN 现有的静态分配语义未考虑任务需求的差异性,统一分配的资源规格与任务实际用量之间的差异引发了大量资源碎片,进一步导致了系统资源利用率降低、集群整体性能下降。

基于 YARN 存在的以上问题,本文设计并实现了作业的自适应资源调度策略,该策略含有全面的节点性能监控、作业特性监控,能有效识别集群优劣势节点,并根据节点优劣性和作业的历史需求信息,弹性地调节作业后续资源容器的分配规格,实现了系统资源的细粒度、动态分配。优化后的系统在完善了 YARN 现有调度语义的同时,能作为子级资源调度方案架构在上层调度器下。基于大数据 benchmark 的性能测试表明,使用本策略优化后的 Hadoop 系统能够有效增加任务并发、减少作业执行时间、提高异构环境中集群的资源利用率。

参考文献:

[1] Lee G, Chun B G, Katz R H. Heterogeneity-aware resource allocation and scheduling in the cloud[C]// Proc of HotCloud,2011:1-5.

[2] Shvachko K,Kuang H,Radia S,et al. The hadoop distributed file system[C]// Proc of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies,2010:1-10.

[3] Kulkarni A P,Khandewal M. Survey on Hadoop and introduction to YARN[J]. International Journal of Emerging Technology and Advanced Engineering,2014,4(5):82-87.

[4] Gupta S,Fritz C,Price B,et al. Throughput scheduler:Learning to schedule on heterogeneous hadoop clusters[C]// Proc of International Conference on Autonomic Computing,2013:159-165.

[5] Rodriguez J M, Mateos C, Zunino A. Energy-efficient job stealing for CPU-intensive processing in mobile devices[J]. Computing,2014,96(2):87-117.

[6] Yao Y,Lin J,Wang J,et al. Admission control in YARN clusters based on dynamic resource reservation[C]// Proc of 2015 IEEE International Symposium on Integrated Network Management,2015:838-841.

[7] Xu Y,Li K,Hu J,et al. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues[J]. Information Sciences, 2014, 270(6): 255-287.

[8] Rendle S, Freudenthaler C, Gantner Z, et al. Bpr: Bayesian personalized ranking from implicit feedback[C]// Proc of the 25th Conference on Uncertainty in Artificial Intelligence, 2009:452-461.

[9] Tan J, Meng X, Zhang L. Coupling task progress for mapreduce resource-aware scheduling[C]// Proc of INFOCOM'13, 2013:1618-1626.
[10] Yang C, Yen C, Tan C, et al. Osprey: Implementing MapReduce-style fault tolerance in a shared-nothing distributed database[C]// Proc of 2010 IEEE 26th International Conference on Data Engineering, 2010:657-668.
[11] Jin Q, Shi Y, Zhao M, et al. The implications from benchmarking three big data systems[C]// Proc of the International Conference on Big Data, 2013:31-38.
[12] Zhou Ao-ying. Data intensive computing-challenges of data management techniques[J]. Communications of China Computer Federation, 2009, 5(7):50-53. (in Chinese)
[13] Liu Wei-ning, Gao Long. Task scheduling strategy based on load balance of clusterin heterogeneous cloud environment [J]. Computer Application, 2013, 33(8):2140-2142. (in Chinese)
[14] Lammel R. Google's MapReduce programming model-revisited[J]. Science of Computer Programming, 2008, 70(1):1-30.
[15] Zhu Q, Agrawal G. Resource provisioning with budget constraints for adaptive applications in cloud environments[C] // Proc of the 19th ACM International Symposium on High Performance Distributed Computing, 2010:304-307.
[16] Polo J, Carrera D, Becerra Y, et al. Performance-driven task co-scheduling for mapreduce environments[C]// Proc of IEEE Network Operations and Management Symposium, 2010:373-380.
[17] Tian C, Zhou H, He Y, et al. A dynamic Mapreduce scheduler for heterogeneous workloads[C]// Proc of the 8th International Conference on Grid and Cooperative Computing, 2009:218-224.
[18] Zaharia M, Konwinski A, Joseph A D, at el. Improving mapreduce performance in heterogeneous environments[C]// Proc of the 8th Usenix Conference on Operating Systems Design and Implementation, 2008:29-42.
[19] Chen Q, Zhang D, Guo M, et al. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment [C]// Proc of 2010 IEEE 10th International Conference on Computer and Information Technology, 2010:2736-2743.
[20] Fritz C. Throughput scheduler: Learning to schedule on heterogeneous Hadoop clusters[C]// Proc of ICAC'13, 2013: 159-165.
[21] Wu Xue-rui. Task scheduling strategy research based on load adaptive in bigdata environment[J]. Huazhong University of Science and Technology, 2013. (in Chinese)
[22] Kc K, Anyanwu K. Scheduling hadoop jobs to meet deadlines[C]// Proc of 2010 IEEE 2nd International Conference on Cloud Computing Technology and Science, 2010: 388-392.
[23] Chatziantoniou D, Tzortzakakis E. ASSET queries: A declarative alternative to MapReduce[C]// Proc of Special Interest Group on Management of Data, 2009:35-41.
[24] Sun H, Cao Y, Hsu W J. Fair and efficient online adaptive scheduling for multiple sets of parallel applications[C]// Proc of 2011 IEEE 17th International Conference on Parallel and Distributed Systems, 2011:64-71.
[25] Raj A, Kaur K, Dutta U, et al. Enhancement of hadoop clusters with virtualization using the capacity scheduler[C]// Proc of 2012 3rd International Conference on Services in Emerging Markets, 2012:50-57.
[26] Ibrahim S, Jin H, Cheng B, at el. Cloudlet: Towards mapreduce implementation on virtual machines[C]// Proc of the 18th ACM International Symposium on High Performance Distributed Computing, 2010:65-66.
[27] Xiong W, Yu Z, Bei Z, et al. A characterization of big data benchmarks[C]// Proc of the International Conference on Big Data, 2013:118-125.

附中文参考文献:

[12] 周傲英. 数据密集型计算——数据管理技术面临的挑战[J]. 计算机学会通讯, 2009, 5(7):50-53.
[13] 刘卫宁, 高龙. 异构云中面向集群负载均衡的任务调度策略[J]. 计算机应用, 2013, 33(8):2140-2142.
[21] 吴雪瑞. 大数据环境下负载自适应的任务调度策略研究[D]. 武汉:华中科技大学, 2013.

作者简介:

潘佳艺(1993-),女,山东禹城人,硕士生,研究方向为大数据。**E-mail**: 15527953855@163.com

PAN Jia-yi, born in 1993, MS candidate, her research interest includes big data.

王芳(1972-),女,河北深县人,博士,教授,CCF 会员(E200005932S),研究方向为海量存储系统、分布式文件系统和大数据处理。**E-mail**: wangfang@mail.hust.edu.cn

WANG Fang, born in 1972, PhD, professor, CCF member(E200005932S), her research interests include massive storage system, distributed file system, and big data processing.

杨静怡(1991-),女,江苏常州人,硕士生,研究方向为大数据。**E-mail**: 493004442@qq.com

YANG Jing-yi, born in 1991, MS candidate, her research interest includes big data.

谭支鹏(1973-),男,湖北巴东人,博士,副教授,研究方向为信息存储技术、大数据存储与管理。**E-mail**: zhipengtan@163.com

TAN Zhi-peng, born in 1973, PhD, associate professor, his research interests include information storage technology, big data storage and management.