Journal of Computer Applications
计算机应用,2014,34(11):3227-3230,3240
ISSN 1001-9081
CODEN JYIIDU
2014-11-10
http://www.joca.cn

文章编号:1001-9081(2014)11-3227-04
doi:10.11772/j.issn.1001-9081.2014.11.3227

基于 Hadoop 的三队列作业调度算法

朱　洁*,赵　红,李雯睿
(南京晓庄学院 数学与信息技术学院,南京 211171)
(*通信作者电子邮箱 wing0909@126.com)

摘　要:Hadoop 集群单队列作业调度会产生短作业等待、资源利用率低的问题;采用多队列调度可兼顾公平、提高执行效率,但会带来手工配置参数、资源互占、算法复杂等问题。针对上述问题,提出三队列作业调度算法,利用区分作业类型、动态调整作业优先级、配置共享资源池、作业抢占等设计,达到平衡作业需求、简化一般作业调度流程、提升并行执行能力的目的。对短作业占比高,各作业占比均衡以及一般作业为主,偶尔出现长、短作业三种情况与先进先出(FIFO)算法进行了对比实验,结果三队列算法的运行时间均比 FIFO 算法要少。实验结果表明,在短作业聚集时,三队列算法的执行效率提升并不显著;但当各种作业并存且分布均衡时,效果很明显,这符合了算法设计时短作业优先、一般作业简化流程、兼顾长作业的初衷,提高了作业整体执行效率。

关键词:Hadoop;作业调度;三队列;共享资源池;作业抢占
中图分类号:TP301.6;TP393.027　　**文献标志码**:A

Three-queue job scheduling algorithm based on Hadoop

ZHU Jie*, ZHAO Hong, LI Wenrui
(*School of Mathematics and Information Technology, Nanjing Xiaozhuang University, Nanjing Jiangsu 211171, China*)

Abstract: Single queue job scheduling algorithm in homogeneous Hadoop cluster causes short jobs waiting and low utilization rate of resources; multi-queue scheduling algorithms solve problems of unfairness and low execution efficiency, but most of them need setting parameters manually, occupy resources each other and are more complex. In order to resolve these problems, a kind of three-queue scheduling algorithm was proposed. The algorithm used job classifications, dynamic priority adjustment, shared resource pool and job preemption to realize fairness, simplify the scheduling flow of normal jobs and improve concurrency. Comparison experiments with First In First Out (FIFO) algorithm were given under three kinds of situations, including that the percentage of short jobs is high, the percentages of all types of jobs are similar, and the general jobs are major with occasional long and short jobs. The proposed algorithm reduced the running time of jobs. The experimental results show that the execution efficiency increase of the proposed algorithm is not obvious when the major jobs are short ones; however, when the assignments of all types of jobs are balanced, the performance is remarkable. This is consistent with the algorithm design rules: prioritizing the short jobs, simplifying the scheduling flow of normal jobs and considering the long jobs, which improves the scheduling performance.

Key words: Hadoop; job scheduling; three-queue; shared resource pool; job preemption

0　引言

云计算作为新兴商业计算模型,已成为业界的研究热点。Hadoop 是 Apache 基金会基于 Google 文件系统、MapReduce 分布式编程模型等核心技术开发的开源云计算平台[1-2],适合并行处理大规模数据。目前互联网领域的 Web 搜索、数据分析、机器学习等都已在 Hadoop 集群上应用。Hadoop 的作业调度算法是否高效直接影响着海量数据的处理效率、集群的资源利用率与平台整体性能。因此,设计高效、公平的调度算法是 Hadoop 研究的核心任务。

现有的 MapReduce 默认调度算法是先入先出(First In First Out, FIFO)算法[3-4],所有作业提交到一个队列中,按照作业的提交顺序执行。该算法简单、易实现、集群开销小,但忽视了不同用户、不同作业的资源需求差异。短作业若排在长作业后,须等待长作业运行完毕,才有机会得以运行。这种不合理影响了集群资源的高效利用与作业整体的执行效率。针对 FIFO 的问题,Facebook 提出公平调度算法[5-7],为每个用户设置了单独的作业池以并行处理作业。新作业到来时重新调度,以确保所有作业能平均分配到资源。算法充分考虑了用户与作业的需求差异,但作业资源配置不够灵活。Yahoo 提出的计算能力调度算法[8]对 FIFO 的改进侧重于提高作业队列的并行能力。多用户共享多队列,限制同一用户占有队列的比例,为各队列配置一定的计算资源。算法考虑了负载均衡,提高了集群资源利用率,但不支持作业抢占,存在资源互占问题。另外,计算能力调度算法与公平调度算法还须手工配置文件参数,用户需要非常熟悉集群系统与需求,配置的

收稿日期:2014-05-16;**修回日期**:2014-07-02。　**基金项目**:国家自然科学基金资助项目(61202136);江苏省高校自然科学研究项目(13KJD520007);南京晓庄学院科学研究项目(2012NXY14,2013NXY99)。
作者简介:朱洁(1979-),女,江苏泰州人,讲师,硕士,主要研究方向:云计算、分布式计算;　赵红(1982-),女,黑龙江哈尔滨人,讲师,博士,主要研究方向:人工智能、分布式计算;　李雯睿(1981-),女,河南开封人,讲师,博士,主要研究方向:云计算、服务计算。

好坏影响着系统的计算性能。随着 Hadoop 的应用要求与环境的多样化，一些新的作业调度算法被提出，这些算法主要在解决数据本地化[9-11]、提高并行效率[12-13]、支持异构环境[14-15]、改善资源调度[16-17]等方面进行了改进。目前，人工智能优化方法逐渐引入作业调度研究领域，如遗传算法、蚁群算法、粒子群算法等。新算法的提出均在一定程度上提高了作业执行效率，但也使作业在执行前需要做更多的预处理，算法变得更加复杂。针对以上传统算法与新算法存在的问题，本文在简化作业流程、减少用户配置、平衡作业需求、灵活配置资源与提高并行计算方面进行研究，提出三队列作业调度算法。经实验证明，本文算法能在一定程度上提升集群作业整体执行效率。

1 算法设计

在默认算法的基础上进行优化与设计：对作业区分类型并赋予优先级，使其能获得及时处理；按作业类型配置队列与资源池以提高并行计算效率；支持作业抢占以防部分作业对资源的长期占用。设计的前提是遵循 Hadoop 的作业执行流程。

1.1 Hadoop 作业执行流程

来自各客户端（JobClient）的作业提交给集群主节点（JobTracker）。JobTracker 同时担当分布式文件系统（Hadoop Distributed File System，HDFS）的名称节点（NameNode），NameNode 管理着文件系统的元数据。作业调度在 JobTracker 上完成，被选取的作业会被划分成若干 Map 任务与 Reduce 任务分配到各从节点（TaskTracker）上执行[18]。TaskTracker 同时担当着 HDFS 的数据节点（DataNode），DataNode 存储着文件的实际数据。TaskTracker 通过心跳包向 JobTracker 请求任务、交换数据信息、报告资源与任务完成情况。作业完成后，JobTracker 将结果返回 JobClient。

1.2 作业区分

作业提交后，在 JobTracker 上入队列等待调度，若短作业与长作业入同一队列将不利于短作业的执行。因此，将作业区分为短作业与一般作业，各自入不同的队列。要预计集群环境下作业执行时间的长短，需要考虑数据本地性、作业启动时 Map 任务的并行度、计算节点上 Map 与 Reduce 任务执行快慢、中间数据提取等诸多因素，这将使区分方法过于复杂。考虑到作业所需的输入数据在作业启动前就已确定，并按照固定的数据块（block）大小进行分片，一个分片对应一个 Map 任务，可以据此简化作业的判定。将 JobTracker 上分片数不超过 *shortJobSplit* 阈值的作业判定为短作业，其余判定为一般作业。

1.3 动态优先级

设置优先级能进一步细化作业差别，根据作业运行情况动态调整优先级可以优化作业的执行效率。优先级变量 *priority* 值越大，作业优先级越高，通过式（1）计算：

$$priority = smallJob \times \omega_1 + progress \times \omega_2 + user \times \omega_3 + initial \times \omega_4 + increment \quad (1)$$

其中：*smallJob* 表示短作业判定值，0 为非短作业，10 为短作业；*progress* 表示作业完成进度，根据 JobTracker 从各 TaskTracker 心跳包中获知的 Map 任务的完成情况进行汇总，利用式（2）将 *progress* 量化为[0,10]。*user* 表示用户身份，对 root、系统用户、普通用户身份进行量化；*initial* 表示 JobClient 对作业预先指定的优先级，取值范围为[0,10]，默认为 0；ω 表示各项在 *priority* 计算中的权重，取值范围为[0,10]，默认为 1；*increment* 表示优先级增量，用于作业被延迟后调高优先级，以利下次调度时能被选取。

$$progress = \mathrm{Round}\left(\frac{M_c}{M+R} \times 10\right) \quad (2)$$

其中：M_c 表示已完成的 Map 任务数，M、R 分别表示该作业 Map 任务与 Reduce 任务总数。作业运行时若被抢占，须计算 *progress* 用于更新 *priority*，以便下次调度时能获得优先执行。

1.4 共享资源池

作业资源池包含各 TaskTracker 资源槽总和，将其划分为一般作业资源池（Job Resource Pool，JRP）与共享资源池（Shared Resource Pool，SRP）。共享资源池主要服务短作业，无短作业时共享给一般作业使用。共享资源池初始值 *SRPinitialSize* 指定为 C_1，C_1 为处理一个数据单元的计算资源大小。队首短作业所需共享资源池大小 C_{SJ} 通过式（3）计算：

$$C_{SJ} = (M+R) \times p \times C_1 \quad (3)$$

其中：M、R 为队首短作业 Map 与 Reduce 任务总数；p 为并行指数，值范围[$1/(M+R)$,1]，p 取 1 时，并行化程度最高。若共享资源池大小 C_{SRP} 无法满足队首短作业资源需求 C_{SJ}，则 C_{SRP} 增加 C_1。若因 JRP 的占用导致 C_{SRP} 无法增加，则判断一般作业资源池大小 C_{JRP} 减少 C_1 能否满足 JRP 中各执行作业的资源需求：

$$C_{JRP} - C_1 \geqslant C_1 \times \sum_i (M_i + R_i) \times p_i \quad (4)$$

若满足，则 C_{SRP} 增加 C_1。如此循环直到 $C_{SRP} \geqslant C_{SJ}$ 为止。因资源总量固定，C_{SRP} 的增加会导致 C_{JRP} 相应减小，资源池结构如图 1 所示。其中，C_S 为计算资源总量。若式（4）无法满足，则等待 JRP 中作业资源释放。若等待时间超过 *smallJobWait* 阈值，则转入 1.6 节介绍的作业抢占机制。

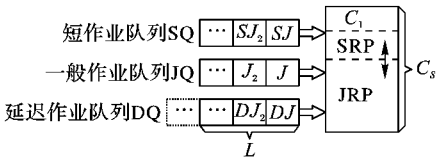


图 1 三阶列与资源池

SRP 的设计保障了短作业的运行。当 JobTracker 中无短作业时，C_{SRP} 恢复为 *SRPinitialSize*，将资源共享给 JRP 中作业使用。若有新的短作业到来，利用式（3）计算该短作业需求，重新进入以上动态调整 C_{SRP} 的过程。每次短作业选取后，优先选择任务数据本地化的计算节点构成 SRP，以减少网络负载。

1.5 作业队列

设立三个作业队列：短作业队列（Small job Queue，SQ）、一般作业队列（Job Queue，JQ）和延迟作业队列（Delay job Queue，DQ）。JobClient 提交的作业若判定为短作业，进入 SQ；否则进入 JQ。JQ 中作业若因资源需求无法满足而暂时无法执行，则认定为延迟作业，进入 DQ。SQ 与 JQ 长度固定，DQ 长度可扩展，初始值为 L。三队列与资源池对应关系如图 1所示。

图 1 中 SQ 队首作业 *SJ*、JQ 队首作业 *J*、DQ 队首作业 *DJ* 通过式（5）判断资源需求是否满足。

$$\begin{cases} C_{SRP} \geqslant C_{SJ} \\ C_{JRP} \geqslant C_J + C_{DJ} \end{cases} \quad (5)$$

其中:C_J、C_{DJ} 表示 JQ、DQ 队首作业资源需求大小,利用式(3)计算。短作业队列队首作业 SJ 若满足式(5),则选取 SJ,利用 SRP 资源执行;若不满足,转入 1.4 节调整 C_{SRP}。一般作业队列队首作业 J 若满足式(5),则获得执行;若无法满足,J 出列并增加其 *increment* 值以调高优先级,进入 DQ。DQ 中存放的作业大多资源需求量大,以长作业居多,优先级被调高利于资源需求满足时能优先于 JQ 中的作业被选取。

延迟队列 DQ 中有作业后,队首作业 DJ 开始参与调度。若此时 J 还未执行,则比较 DJ 与 J 的优先级,优先级高且资源需求能满足的队首作业获得执行;若此时 J 已执行,DJ 也能得到足够的计算资源,则 DJ 获得执行。若资源需求无法满足,则比较 DJ 与 J_2 的优先级(J_2 为一般作业队列队首作业的后续作业)。若 DJ 优先级较高且 C_{JRP} 大于 DJ 所需资源,则等待 J 执行完毕后执行 DJ。在以上过程中,因优先级低或资源需求无法满足的作业,将被调高优先级,重入延迟队列。采用直接重入延迟队列这种简单化处理方式,可使作业无须等待,资源利用率更高。比较 J_2 的原因是:若 DJ 资源需求较大,始终得不到足够的资源,即使优先级非常高,也很难被选取,应尽可能地给予延迟作业执行的机会。

延迟队列 DQ 中无作业时,将一般作业队列 JQ 中距离队首最近的可能延迟作业转入 DQ。可能延迟作业为分片数超过 *delayJobSplit* 阈值的作业。若 JQ 中没有可能延迟作业,将 J_2 转入 DQ。如此便可在无延迟作业时利用 DQ 执行一般作业,提高并行执行效率。

延迟队列长度 L_{DQ} 根据作业入列的实际情况进行调整。DQ 中作业数达到 L 时,JQ 不再接受新作业。若 JQ 中有作业要进入 DQ,L_{DQ} 加 1。DQ 中队首作业执行完毕,L_{DQ} 减 1。DQ 中作业数为 $L-1$ 时,JQ 开始接受新作业。为了提高集群系统作业执行的整体效率,设置了延迟队列。为了保障延迟作业的执行,使用了动态优先级与 SRP 的共享。若延迟队列中作业过多,停止 JQ 的作业接收,使堆积的延迟作业能得到及时地处理。

1.6　作业抢占

有两种情况需要增加共享资源池 SRP 的大小:1)队首短作业资源需求无法满足;2)因暂时无短作业,将 SRP 共享后,新的短作业到来且 $C_{SJ} > C_1$。在调整 C_{SRP} 的过程中,若因 JRP 中正在执行的作业占用,导致无法继续增加 C_{SRP},则队首短作业须等待。若等待时间超过 *smallJobWait* 阈值且队首一般作业 J 的执行进度不足 50%,则 J 被抢占。判断条件如式(6):

$$(C_{JRP} - C_1) < (C_J + C_{DJ}) \&\& (T_{SJ} > smallJobWait \&\& process_J \leqslant 5) \quad (6)$$

若条件满足,则 J 终止,资源释放以利短作业执行。若 J 的执行进度超过 50%,则短作业继续等待,C_{SRP} 恢复到初始值 *SRPinitialSize*,以加速 JRP 中作业的完成。直到 JRP 中有资源释放且满足队首短作业需求时,短作业得以执行。

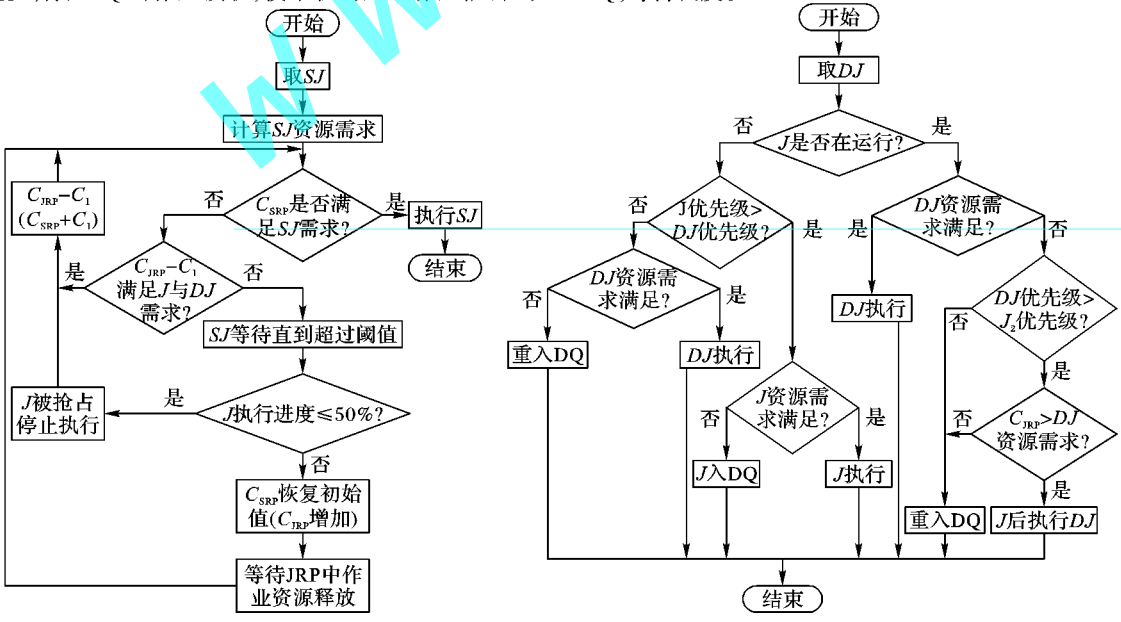
延迟作业是因资源需求暂时无法满足而等待再次调度的作业,若有机会得以执行,不应被抢占。在上述设计中,若 J 未被抢占,延迟作业会获得 SRP 资源的共享;若 J 被抢占,也会得到 J 释放的计算资源。不管抢占是否发生,都会加速延迟作业的执行。

2　算法实现

在 Hadoop 作业流程中实现算法设计思想。JobClient 作业提交到 JobTracker 后,对作业进行判定,短作业入 SQ,一般作业入 JQ。短作业流程如图 2(a)所示。

为保障短作业执行,从资源池中划分出 SRP,当其不能满足短作业需求时,可逐步增加 SRP 大小。暂时无短作业时,SRP 可共享给一般作业使用。若因一般作业的资源占用而无法增加 SRP 时,根据一般作业的执行进度,决定是否抢占一般作业。

一般作业在集群中占有较大比例,简化 JQ 中一般作业的流程以降低算法复杂度。JQ 队首作业资源需求能得到满足,则获得执行;若无法满足,则调高优先级,成为延迟作业进入 DQ,等待调度。



(a) 短作业流程　　(b) 延迟作业流程

图 2　短作业和延迟作业流程

延迟作业优先级较高,使用 JRP 中的计算资源。资源需求能满足的作业会被执行,无法满足或优先级低的作业将被调高优先级,重新进入 DQ 排队,流程如图 2(b)所示。对于其中资源需求量非常大的作业,优先级若高于 JQ 队首作业 J 的后续作业 J_2,J 完成后若 JRP 能满足该作业资源需求,则会被执行。若仍不能执行,则在无短作业时,SRP 共享后,JRP 大小为 $C_S - C_1$ 时会被执行,此时资源供给已接近最大值。

3 实验与分析

选取 5 台 PC(配置为:CPU 双核 2.4 GHz,内存 4 GB,硬盘 320 GB,CentOS Server 操作系统,Hadoop 0.21.0 版本)搭建 Hadoop 集群,其中:1 台机器作为 JobTracker,其余 4 台作为 TaskTracker。Hadoop 默认配置每个计算节点拥有 2 个资源槽。测试程序来自 Hadoop Hive 项目提供的基准测试程序 WordCount。选择 WordCount 作业,因其对于 CPU、I/O、存储都有要求,属于资源需求均衡型作业。短作业数据文件大小在 1 个 block(64 MB)以内,一般作业数据大小大部分为 64 MB ~ 2 GB,其中可能延迟的长作业数据大小为 10 GB。

在作业数固定、提交顺序随机的前提下,比较三种情况下三队列调度算法与 FIFO 算法的作业执行效率。实验 1:短作业占 90%,一般作业占 10%,无长作业,结果如图 3(a)所示。实验 2:短作业、长作业各占 30%,其余一般作业占 40%,结果如图 3(b)所示。实验 3:短作业、长作业各占 10%,其余一般作业占 80%,结果如图 3(c)所示。未设计长作业占比高的实验,因为长作业较多的情况下,治本的方法是增加资源供给,而不是增加算法复杂度。

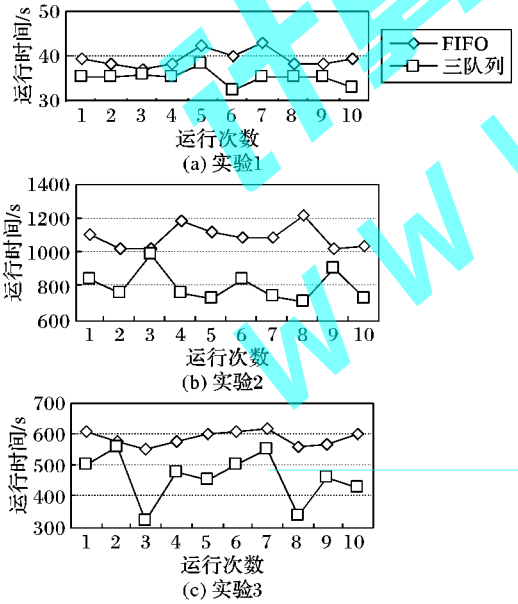


图 3 三种情况下算法运行时间对比

分析实验 1 的结果得出三队列算法在运行时间上较 FIFO 算法缩短 10.97%。一般作业不管何时提交,与其后提交的短作业因分属不同的队列而并行执行,使作业总运行时间得以减少。实验 2 的设计接近于实际服务应用,各作业比例相当,三队列算法的运行时间较 FIFO 算法缩短近26.78%。个别次的运行时间接近 FIFO 算法,因为作业提交顺序是随机的,长作业若晚来到,算法的优化作用并不明显。实验 3 以一般作业为主,偶尔出现的长作业或短作业会与一般作业并行,三队列算法的运行时间较 FIFO 算法缩短 21.87%。若长作业晚到来,优化作用仍不明显。

算法所能缩短的运行时间与资源池的设置、提交作业的设计均有关,通过实验结果能够得出:在短作业聚集时,三队列算法的执行效率提升并不显著;当各种作业并存且分布均衡时,效果最明显,这符合了算法设计时短作业优先、一般作业简化流程、兼顾长作业的初衷,提高了作业整体执行效率。

4 结语

三队列算法在 FIFO 算法的基础上进行优化,融入并行执行、兼顾公平的设计思想,以简单、公平、高效为原则,无须预先设置,实现了作业区分,使得短作业在各种作业共存时能得到单独处理,不作无谓等待;根据作业的类型与执行情况动态调整作业优先级,以优先执行短作业,保障了延迟作业;设置 SRP 确保短作业的资源供给,并在无短作业时共享,使资源配置更加灵活,提高了整体性能;为各种作业分别设置队列,简化一般作业流程,在资源满足的情况下优先选择延迟作业,提高了并行能力;支持作业抢占,保障了短作业与延迟作业的执行。

实验结果证明,算法在各作业并存的情况下,能有效地提高作业执行效率。对于 Hadoop 同构环境下的作业调度可提供一定的借鉴。下一步可对作业类型细分,自适应以支持作业多样性,并继续改进并行机制,以使作业调度更精确与高效。

参考文献:

[1] Cloud computing[EB/OL]. [2013-03-10]. http://en.wikip-edia.org/wiki/Cloud_computing.

[2] DEAN J, GHEMAWAT S. MapReduce: a flexible data processing tool[J]. Communications of the ACM, 2010, 53(1): 72 - 77.

[3] MapReduce[EB/OL]. [2013-03-10]. http://zh.wikipedia.o-rg/zh-cn/MapReduce.

[4] ZAHARIA M. Job scheduling with the fair and capacity schedulers [EB/OL]. [2013-05-10]. http://www.cs.berk-eley.edu/~matei/talks/2009/hadoop_summit_fair_sched-uler.pdf.

[5] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107 - 113.

[6] ZAHARIA M, BORTHAKUR D, SARMA J S, *et al*. Job scheduling for multi-user MapReduce clusters, 55[R]. Berkeley: University of California, 2009.

[7] Fair scheduler guide[EB/OL]. [2013-04-08]. http://had-oop.apache.org/docs/r1.2.1/Fair_Scheduler.html.

[8] Capacity scheduler guide[EB/OL]. [2013-04-08]. http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html.

[9] GUO Z, FOX G, ZHOU M. Investigation of data locality and fairness in MapReduce[C]// Proceedings of the 3rd International Workshop on MapReduce and its Applications. New York: ACM Press, 2012: 25 - 32.

[10] ZAHARIA M, BORTHAKUR D, SARMA J S, *et al*. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]// EuroSys 2010: Proceedings of the 5th European Conference on Computer Systems. New York: ACM Press, 2010: 265 - 278.

(下转第 3240 页)

[4] ZHANG X, YAN Y, HE K. Latency metric: an experimental method for measuring and evaluating parallel program and architecture scalability [J]. Journal of Parallel and Distributed Computing, 1994, 22(3): 392 – 410.

[5] WU X, LI W. Performance models for scalable cluster computing [J]. Journal of Systems Architecture, 1997, 44(3): 189 – 205.

[6] CHEN J, LI X. Near-optimal scalability: a practical scalability metric [J]. Chinese Journal of Computers, 2001, 24(2): 179 – 182. (陈军, 李晓梅. 近优可扩展性:一种实用的可扩展性度量 [J]. 计算机学报, 2001, 24(2): 179 – 182.)

[7] WANG Y, YANG X. A more effective scalability model for parallel system [J]. Chinese Journal of Computers, 2001, 24(1): 84 – 90. (王与力, 杨晓东. 一种更有效的并行系统可扩展性模型 [J]. 计算机学报, 2001, 24(1): 84 – 90.)

[8] CHI L, LIU J, HU Q. Evaluation and test for scalability of numerical parallel computation [J]. Journal of Computer Research and Development, 2005, 42(6): 1073 – 1078. (迟利华, 刘杰, 胡庆丰. 数值并行计算可扩展性评价与测试 [J]. 计算机研究与发展, 2005, 42(6): 1073 – 1078.)

[9] BOSQUE J L, ROBLES O D, TOHARIA P. Evaluating scalability in heterogeneous systems [J]. Journal of Supercomputer, 2011, 58(3): 367 – 375.

[10] CHEN Y, SUN X, WU M. Algorithm-system scalability of heterogeneous computing [J]. Journal of Parallel and Distributed Computing, 2008, 68(11): 1403 – 1412.

[11] YANG X. Sixty years of parallel computing [J]. Computer Engineering and Science, 2012, 34(8): 1 – 10. (杨学军. 并行计算六十年[J]. 计算机工程与科学, 2012, 34(8): 1 – 10.)

[12] WANG Z, YANG X. Metrics of measuring parallel computing systems [J]. Computer Engineering and Science, 2010, 32(10): 44 – 48. (王之元, 杨学军. 并行计算系统度量指标综述 [J]. 计算机工程与科学, 2010, 32(10): 44 – 48.)

[13] YANG X, WANG Z, XUE J. The reliability wall for exascale supercomputing [J]. IEEE Transactions on Computers, 2012, 61(6): 767 – 779.

[14] XIONG H, ZENG G, WU C, *et al.* Relationships between latency scalability and execution time [J]. Journal of Computer Applications, 2014, 34(3): 663 – 667. (熊焕亮, 曾国荪, 吴沧海, 等. 延迟可扩展性与并行执行时间的关系[J]. 计算机应用, 2014, 34(3): 663 – 667.)

[15] SUN X, NI L. Scalable problems and memory-bounded speedup [J]. Journal of Parallel and Distributed Computing, 1993, 19(1): 27 – 37.

[16] YANG X, DU J, WANG Z. An effective speedup metric for measuring productivity in large-scale parallel computer systems [J]. Journal of Supercomputing, 2011, 56(2): 164 – 181.

[17] XIONG H, ZENG G, ZENG Y. A novel scalability metric about iso-area of performance for parallel computing [J]. Journal of Supercomputing, 2014, 68(2): 652 – 671.

[18] CAO J, ZENG C, NIU J, *et al.* Availability-aware scheduling method for parallel task in cloud environment[J]. Journal of Computer Research and Development, 2013, 50(7): 1563 – 1572. (曹洁, 曾国荪, 钮俊, 等. 云环境下可用性感知的并行任务调度方法[J]. 计算机研究与发展, 2013, 50(7): 1563 – 1572.)

[19] LI X, ZHANG X. Energy-aware resource management mechanism for virtualized cloud datacenter [J]. Journal of Computer Applications, 2013, 33(12): 3586 – 3590. (李小六, 张曦煌. 虚拟化云计算数据中心能量感知资源分配机制[J]. 计算机应用, 2013, 33(12): 3586 – 3590.)

[20] DUBOC L, ROSENBLUM D S, WICKS T. A framework for characterization and analysis of software system scalability [C]// Proceedings of the 6th Joint Meeting of the European Software Engineering. New York: ACM Press, 2007: 375 – 384.

[21] HAO S, ZENG G, TAN Y. Scalability analysis of heterogeneous computing based on computation task and architecture to match [J]. Acta Electronica Sinica, 2010, 38(11): 2585 – 2589. (郝水侠, 曾国荪, 谭一鸣. 计算任务与体系结构匹配的异构计算可扩展性分析[J]. 电子学报, 2010, 38(11): 2585 – 2589.)

(上接第 3230 页)

[11] KE H, YANG Q, WANG L, *et al.* Improved delay-scheduler algorithm in homogeneous Hadoop cluster[J]. Application Research of Computers, 2013, 30(5): 1397 – 1401. (柯何杨, 杨群, 王立松, 等. 同构 Hadoop 集群环境下改进的延迟调度算法[J]. 计算机应用研究, 2013, 30(5): 1397 – 1401.)

[12] JIN W, WANG C. Iteration MapReduce framework for evolution algorithm[J]. Journal of Computer Applications, 2013, 33(12): 3591 – 3595. (金伟健, 王春枝. 适于进化算法的迭代式 MapReduce 框架[J]. 计算机应用, 2013, 33(12): 3591 – 3595.)

[13] ZHANG X, GONG K, ZHAO G. Parallel *k* - medoids algorithm based on MapReduce[J]. Journal of Computer Applications, 2013, 33(4): 1023 – 1025. (张雪萍, 龚康莉, 赵广才. 基于 MapReduce 的 *k*-medoids 并行算法[J]. 计算机应用, 2013, 33(4): 1023 – 1025.)

[14] ZAHARIA M, KONWINSKI A, JOSEPH A D, *et al.* Improving MapReduce performance in heterogeneous environment[C]// Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX, 2008: 29 – 42.

[15] LI L, TANG Z, LI R. New improvement of the Hadoop relevant data locality scheduling algorithm based on LATE[J]. Computer Science, 2011, 38(11): 67 – 70. (李丽英, 唐卓, 李仁发. 基于 LATE 的 Hadoop 数据局部性改进调度算法[J]. 计算机科学, 2011, 38(11): 67 – 70.)

[16] POLO J, CARRERA D, BECERRA Y, *et al.* Performance-driven task co-scheduling for MapReduce environments[C]// Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium. Piscataway: IEEE Press, 2010: 373 – 380.

[17] DENG C, FAN T, GAO F. A resource scheduler algorithm based on statistical optimization under Hadoop[J]. Application Research of Computers, 2013, 30(2): 417 – 422. (邓传华, 范通让, 高峰. Hadoop 下基于统计最优的资源调度算法[J]. 计算机应用研究, 2013, 30(2): 417 – 422.)

[18] Hadoop MapReduce[EB/OL]. [2013-04-12]. http://wiki.apache.org/hadoop/HadoopMapReduce.