CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 35 卷第 10 期 2013 年 10 月
Vol. 35, No. 10, Oct. 2013

文章编号:1007-130X(2013)10-0125-06

Hadoop 平台中空闲时间调度器的设计与实现*

杨 浩,滕 飞,李天瑞,李 墨
(西南交通大学信息科学与技术学院,四川 成都 610031)

摘 要:Hadoop 作为开源的云计算平台,被广泛应用于自然语言处理、机器学习、大规模图像处理等领域。随着云计算与各行业广泛而深入的结合,多样化的服务对于时效性要求越来越高。现有的 Hadoop 调度器多关注于缩短响应时间,而非满足作业的时限要求。为了提高集群处理硬实时作业的性能,设计并实现了一种基于空闲时间的硬实时调度器 LSS。在调度过程中,LSS 动态估算作业的空闲时间,并据此实时更新作业队列中作业的优先级顺序。实验结果表明 LSS 能够有效地提高集群处理硬实时作业的成功率。

关键词:云计算;实时作业;调度;空闲时间
中图分类号:TP338.8 **文献标志码**:A
doi:10.3969/j.issn.1007-130X.2013.10.016

Design and implementation of a least spare time scheduler for Hadoop

YANG Hao, TENG Fei, LI Tian-rui, LI Zhao
(School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: As an open source platform of cloud computing, Hadoop is widely used in many fields, such as natural language processing, machine learning and large-scale image processing. With the increase of the types of cloud services, the real-time requirement is strengthened by cloud users. Most existing schedulers are designed to shorten the response time which cannot guarantee a specific deadline. Least Spare-time Scheduler (LSS) is designed and implemented to improve the performance of hard real-time jobs in Hadoop. The spare time is estimated dynamically and the LSS updates the job priority of the job queue in real-time. Experimental results show that the LSS can improve the success ratio of the cluster dealing with hard real-time jobs.

Key words: Hadoop; real-time jobs; scheduling; spare time

1 引言

快速发展的计算机技术、网络技术和人机交互技术等,让诸如天文、军事、社交等领域的数据生成和采集更为便利和快捷,从而使得数据规模不断增长,形成了大数据。处理大数据并从中提取有用的信息,需要良好的并行计算架构。MapReduce 是云计算中很流行的并行计算框架之一[1],其优势在于隐藏了数据并行处理的细节,让用户更专注于数据的处理逻辑[2]。MapReduce 已被广泛用于自然语言处理、机器学习、大规模图像处理等领域[3]。

Hadoop 是 Apache 软件基金提供的基于 MapReduce 的开源云计算平台。Hadoop 集群由主节点和多个从节点构成[4]。主节点用于接收用户提交的作业,并将作业的 Map 任务和 Reduce 任务平均分配到各从节点上运行,以达到并行计算的目的。Hadoop 将从节点上的计算资源抽象成插槽,

* 收稿日期:2013-06-15;修回日期:2013-09-10
基金项目:国家自然科学基金资助项目(61202043,61175047);中央高校基本科研业务费专项资金(SWJTU11ZT08,SWJTU12CX098)
通讯地址:610031 四川省成都市西南交通大学信息科学与技术学院 滕飞
Address: School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, Sichuan, P. R. China

每一个插槽在同一时刻可处理一个 Map 任务或者 Reduce 任务。Hadoop 上调度器的功能是将空闲插槽分配给不同任务,因此调度算法的好坏直接关系到 Hadoop 集群的整体性能和资源的利用情况[5]。

实时作业分为软实时作业和硬实时作业。软实时作业是指作业的响应时间可以超过其截止时间;而硬实时作业是指作业的响应时间不允许超过其截止时间。硬实时作业是刚性的,超过截止时间的作业将不再有任何意义。Hadoop 中现有的调度算法大多是针对集群的利用率、作业响应速度等方面设计的,而针对硬实时作业对截止时间要求的调度算法比较少。

本文研究 Hadoop 上的作业调度模型,设计并实现了一种基于作业空闲时间的硬实时调度器 LSS。为了提高集群在处理硬实时作业时的成功率,减轻集群的负载,LSS 设计过程中采用了以下四个机制:

(1)LSS 利用空闲时间来描述作业的优先级。空闲时间是指作业距离其截止时间前还可以延迟的最大时间。空闲时间越小的作业,其紧急程度越大,作业的优先级也会越高。

(2)LSS 动态预测作业的空闲时间,并随着时间的推移,LSS 实时更新预测结果。LSS 是根据作业运行过程中产生的数据对空闲时间进行预测的,实时更新有效地保证了预测结果的正确性。

(3)LSS 实时更新作业在作业队列中的优先级顺序,将空闲时间比较小的作业移到队列前面,当有资源空闲时,空闲时间小的作业就会优先执行,这样有效地保证了集群中时限作业的成功执行。

(4)当预测出当前资源无法保证作业在截止期前完成时,LSS 会提前将这些无法成功执行的作业结束。这样能有效地减轻集群的负载,同时增加其他时限作业成功执行的可能性。

2 相关工作

Hadoop 默认调度器的调度算法是 FIFO,适合批处理作业,但是该调度器在多用户并存、资源利用率等方面存在不足,因此 Hadoop 中出现了其他调度器,并分别从不同角度对作业调度进行了优化。公平调度器[6]和容量调度器用以解决多用户多队列的调度问题。LATE 调度器[7]针对集群的异构进行了优化。MFS[5]使用需求向量来描述作业对各类资源的需求大小,满足了不同作业对资源的不同需求。许丞等[8]将 JobTracker 上的资源管理和任务监控分布到不同节点上,降低了主节点的负载。这些调度器对提高 Hadoop 集群的资源管理和任务调度都取得了一定成果,但是未能对作业的实时性进行优化。

随着用户对云计算中作业时效性的要求越来越高,Hadoop 平台中也出现了一些实时调度器。Delay Scheduler[6]、DSFS[9]等根据数据本地性对作业的执行进行了优化。魏晓辉[10]设计了一种基于资源预测的 Delay 调度算法,解决了 Delay 调度算法中不合理等待问题。宁文瑜等人[11]设计的自适应延迟调度算法,动态调整作业的等待时间,优化了公平性和数据本地性之间的平衡。Deadline Constraint Scheduler[4]在作业提交时判断当前集群能否满足作业的时限要求,如果无法满足,则结束该作业,这样有效地保证了其他作业的成功执行。董西成等人[12]设计了一种能同时处理实时作业和非实时作业的调度器。这些调度器虽然缩短了作业的整体响应时间,但是无法保证硬实时作业的时限要求。为提高集群在处理硬实时作业时的整体性能,本文设计并实现了 LSS 调度器。该调度器不仅可以保证硬实时作业的时限要求,而且还可以有效地减轻集群的负载。

3 LSS 调度器

3.1 LSS 中空闲时间估计

LSS 的关键在于空闲时间的估算,本小节将介绍其空闲时间的估算方法。

(1)空闲时间定义。

空闲时间是指作业距离其截止时间前还可以延迟的最大时间。式(1)为作业空闲时间的计算公式。

$$T_{Spare} = T_{DeadLine} - T_{Now} - T_{Remaining} \quad (1)$$

其中,T_{Spare} 表示作业的空闲时间;$T_{DeadLine}$ 表示该作业的截止时间;T_{Now} 表示当前时间;$T_{Remaining}$ 表示该作业剩余执行时间,即如果系统中只有该作业执行,作业还需要执行的时间。

(2)剩余时间估计。

MapReduce 分为 Map 阶段和 Reduce 阶段。在 Map 阶段,如果任务分片设置合理,在误差允许的范围内,每个 Map 任务的执行时间是一样的。而对于 Reduce 阶段,由于其执行时间难以估计,可以粗略将 Reduce 任务的执行时间等效于 Map 任务的执行时间,即所有任务的执行时间是相同

的。本文用已经执行完的任务的平均执行时间来预测未完成的任务的执行时间。式(2)为任务的平均执行时间预测方法。

$$T_{\text{AvgTask}}=\frac{\sum T_{\text{FinishedTask}}}{N_{\text{FinishedTask}}} \tag{2}$$

其中，T_{AvgTask} 表示任务的平均执行时间，$\sum T_{\text{FinishedTask}}$ 表示所有执行完的任务的执行时间之和，$N_{\text{FinishedTask}}$ 表示已经执行完的任务的数目。

没有执行完的任务分为 Running 任务(正在执行,但没有执行完的任务)和 Pending 任务(已经准备好,但没有开始执行的任务),而其中 Running 任务已经执行了一段时间,剩余时间中应该不包括这部分。同时,考虑作业中未执行完的子任务数目、子任务的并行度,以及 Map 任务和 Reduce 任务的并行度不一样等因素,用式(3)计算 $T_{\text{Remaining}}$。

$$T_{\text{Remaining}}=\left(\left\lceil\frac{N_{\text{unFinishedMapTask}}}{N_{\text{MapSlot}}}\right\rceil+\left\lceil\frac{N_{\text{unFinishedReduceTask}}}{N_{\text{ReduceSlot}}}\right\rceil\right)\times T_{\text{AvgTask}}-\frac{\sum T_{\text{RunningMapTask}}}{N_{\text{MapSlot}}}-\frac{\sum T_{\text{RunningReduceTask}}}{N_{\text{ReduceSlot}}} \tag{3}$$

其中，$N_{\text{unFinishedMapTask}}$ 表示未执行完的 Map 任务的数目，N_{MapSlot} 表示集群中 Map 插槽的槽数，$N_{\text{unFinishedReduceTask}}$ 表示未执行完的 Reduce 任务的数目，$N_{\text{ReduceSlot}}$ 表示集群中 Reduce 插槽的槽数,符号 $\lceil\ \rceil$ 表示向上取整，$\sum T_{\text{RunningMapTask}}$ 表示正在运行的 Map 任务已经执行的时间之和，$\sum T_{\text{RunningReduceTask}}$ 表示正在运行的 Reduce 任务已经执行的时间之和。

(3)LSS 中空闲时间估算。

将(3)中得到的 $T_{\text{Remaining}}$ 代入式(1),即可得到 T_{Spare},如式(4)所示。

$$T_{\text{Spare}}=T_{\text{DeadLine}}-T_{\text{Now}}-T_{\text{Remaining}}=T_{\text{DeadLine}}-T_{\text{Now}}-\left(\left(\left\lceil\frac{N_{\text{unFinishedMapTask}}}{N_{\text{MapSlot}}}\right\rceil+\left\lceil\frac{N_{\text{unFinishedReduceTask}}}{N_{\text{ReduceSlot}}}\right\rceil\right)\times T_{\text{AvgTask}}-\frac{\sum T_{\text{RunningMapTask}}}{N_{\text{MapSlot}}}-\frac{\sum T_{\text{RunningReduceTask}}}{N_{\text{ReduceSlot}}}\right) \tag{4}$$

任务提交时无法计算 T_{AvgTask}。本文采用如下方式处理:在作业提交时，T_{Spare} 用式(5)来计算;而当作业有任务执行完时,用式(4)来计算 T_{Spare}。

$$T_{\text{Spare}}=T_{\text{DeadLine}}-T_{\text{Now}} \tag{5}$$

3.2　LSS 调度器设计

(1)预测并提前结束不能完成的作业。

对于传统的硬实时作业调度器,当作业的截止时间大于当前时间时,就结束该作业。此调度过程没有考虑作业不能成功执行的其它情况。当 $T_{\text{Spare}}<0$ 时,即使让作业继续执行,该作业也不会在截止时间之前完成,所以 LSS 还增加了另外一个结束作业的条件:当 $T_{\text{Spare}}<0$ 时,结束该作业。由于估算出来的 T_{Spare} 有时会偏小,为了避免将能正常执行的作业结束,所以在其上加了一个阈值 T_{AvgTask},即当 $T_{\text{Spare}}+T_{\text{AvgTask}}<0$ 时,结束该作业。

(2)LSS 调度流程。

Hadoop 集群上的节点分为主节点和从节点,是一种典型的主从式(Master/Slave)架构[13]。和调度器相关的守护进程为主节点上的 JobTracker 和从节点上的 TaskTracker。JobTracker 守护进程负责为作业队列分配资源,而 TaskTracker 负责 Map 任务或者 Reduce 任务在从节点中的插槽上运行。当从节点上有插槽空闲时,则 TaskTracker 通过心跳机制将该信息发送给 JobTracker,JobTracker 通过 *scheduleTasks*() 函数为 TaskTracker 对应的节点分配资源。算法 1 描述了 LSS 中的该过程,本算法包括以下几个步骤:(1)重新计算作业队列中各作业的平均时间和空闲时间;(2)将超过截止时间的作业或者空闲时间小于给定阈值的作业结束;(3)根据空闲时间对作业队列排序;(4)获取 TaskTracker 对应从节点的空闲插槽;(5)为刚提交的作业分配一个 Map 插槽;(6)按照作业队列中的顺序为作业分配插槽。

算法 1　*scheduleTasks*()

输入:*jobQueue* 为作业队列,*t* 为 TaskTracker。

输出:*assignedTasks* 为分配的任务列表。

开始:

```
1.  synchronized (jobQueue);//同步作业队列
2.  calAvgTime (jobQueue);//计算平均时间
3.  calSpareTime (jobQueue);//计算空闲时间
4.  killJobByDeadline (jobQueue);
5.  killJobBySpareTime (jobQueue);
6.  sortBySpareTime (jobQueue);//排序
7.  synchronized (t);//同步 t
8.  generate (freeMapSlotsList, t);/*获取 t 中空闲 Map 插槽*/
9.  generate (freeReduceSlotsList, t);/*空闲 Reduce 插槽*/
10.  FOR job in jobQueue DO
11.    IF numFreeMapSlots==0 THEN
12.      BREAK
13.    END IF
14.    IF job.getNumfinishedTasks()+job.get-
```

```
    NumRunningTasks()==0 THEN
15.   assignMapTask(job.getUnassignedMapTasks
      (),freeMapSlotsList, 1);
16.   assignedTasks.append(Maps);
17.   END IF
18. END FOR
19. FOR job in jobQueue DO
20.   IF numFreeMapSlots==0 THEN
21.     BREAK
22.   END IF
23.   assignMapTask(job.getUnassignedMapTasks
      (),freeMapSlotsList, numFreeMapSlots);
24.   assignedTasks.append(Maps);
25.   END FOR
26. FOR job in jobQueue DO
27.   IF numFreeReduceSlots== 0 THEN
28.     BREAK;
29.   END IF
30.   assignReduceTask(job.getUnassignedReduc-
      eTasks(), freeReduceSlotsList, numFreeRe-
      duceSlots);
31.   assignedTasks.append(Reduces);
32. END FOR
```

结束。

3.3 LSS 调度器实现

本文在 Hadoop 中实现了 LSS 调度器。LSS 的实现主要包括以下几个部分:(1)增加作业属性:每一个实时任务都有自己的截止时间,需要给作业添加截止时间属性;(2)继承抽象类 TaskScheduler:主要工作是修改其中分配任务的方法,LSS 按照算法 1 修改了 TaskScheduler 中的 scheduleTasks()方法;(3)集群配置:将改写好的调度器打成 Jar 包,放在主节点 Hadoop 根目录下的 lib 文件夹下,同时在配置文件 mapred-site.xml 中添加属性“mapred.jobtracker.taskScheduler”,并将其值赋为改写后的 TaskScheduler 类“org.apache.hadoop.mapred.LLScheduler”。重启集群后,集群采用的调度器就为 LSS。

4 实验与结果分析

4.1 集群配置

本文所使用的集群环境如表 1 所示。Hadoop 版本为 Hadoop-1.1.0。集群包括六个节点:一个主节点和五个从节点,每个从节点包括两个 Map 插槽和一个 Reduce 插槽。

Table 1 Experimental configuration

表 1 实验环境

参数	值
机器配置	DELL(Xeon E5506/12GHz/600GB)
Hadoop 版本	Hadoop-1.1.0
JDK	Jdk1.7.0
Hadoop 配置	一个主节点、五个从节点

4.2 实验方案

实时系统中常用工作流描述提交的作业,同一工作流中提交同一类型的作业。工作流用 $W_i=(\lambda_i,D_i,S_i,M_i,R_i)$ 表示,其中 λ_i 表示作业的到达率,作业到达时间间隔服从 $1/\lambda_i$ 的泊松分布,零时刻没有作业提交,如果 $\lambda_i=1/500$,则该工作流中作业的到达时刻可能为 482、1 019、1 527、……,如图 1 所示;D_i 表示作业的相对截止时间,作业的截止时间为作业提交时间加上 D_i;S_i 表示作业处理的输入数据大小;M_i 表示作业中 Map 任务的数目;R_i 表示作业中 Reduce 任务的数目。

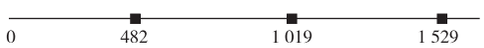

Figure 1 The possible reach time of jobs when $\lambda_i=1/500$

图 1 $\lambda_i=1/500$ 时工作流中作业可能到达的时刻

本文设计了两组实验,一组为轻负载,一组为重负载。每一组实验有三个工作流,并分别在默认调度器和 LSS 上做了实验。实验 1 为轻负载,该实验中的 D_i 为其作业平均到达时间间隔,即 $D_i=1/\lambda_i$;实验 2 为重负载,该实验中的 D_i 小于其作业平均到达时间间隔,即 $D_i<1/\lambda_i$。实验方案如表 2 所示。WordCount 为 Hadoop 的基准测试用例,本文中的六个工作流选择的作业均是 WordCount。

Table 2 Experimental scheme

表 2 实验方案

参数	值
实验 1	$W_1=(1/500,500,3.35\,GB,108,1)$ $W_2=(1/300,300,1.21\,GB,39,1)$ $W_3=(1/200,200,623\,MB,20,1)$
实验 2	$W_4=(1/500,400,3.35\,GB,108,1)$ $W_5=(1/300,240,1.21\,GB,39,1)$ $W_6=(1/200,160,623\,MB,20,1)$
调度器	Default,LSS

4.3 实验结果与实验分析

图 2a 和图 2b 分别为实验 1 中默认调度器和 LSS 中不同工作流的资源占用情况。从图中可以看出,LSS 调度器中工作流对资源的抢占比 FIFO 激烈。这是因为 LSS 动态更新了作业队列中作业的优先级顺序,而 FIFO 调度器中作业的优先级顺

序是一直不变的。在LSS中,虽然作业之间会相互抢占,但抢占消耗的主要是JobTracker上的资源,对从节点的影响不大,因此作业队列中作业优先级顺序的动态更新对作业在从节点中的执行影响不大。

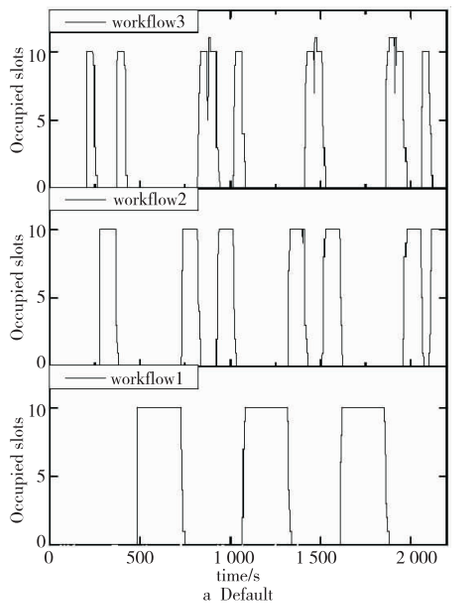

Figure 2　Occupied slots of the default scheduler under experiment 1

图2　实验1默认调度器中不同工作流的资源占用情况

图3为实验1和实验2中默认调度器和LSS调度器的成功率。从图中可以看出,不管是在轻负载还是在重负载的情况下,LSS调度器的成功率均高于默认调度器,说明LSS能更好地处理硬实时作业。因为LSS实时更新了作业队列中的作业的优先级顺序,空闲时间小的作业的执行得到了保证;而且LSS提前将不能成功执行的作业结束了,给其他作业留出了更多的资源,保证了其他作业成功执行的可能性。但是,默认调度器只采用了先进先出的设计思想,没有对实时作业做任何优化。因此,在两种情况下,LSS调度器的成功率均高于默认调度器。

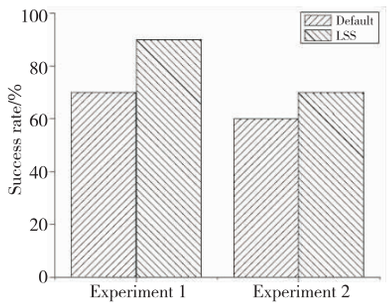

Figure 3　Success ratios of experiment 1 and experiment 2

图3　实验1和实验2中的成功率

图4a和图4b分别为实验1、实验2中不同工作流中作业的平均响应时间。从图中可以看出,工作流1中默认调度器的效果比LSS的好。这是因为LSS倾向于保障时限要求更高的工作流的执行,当优先级更高的工作流2和工作流3有计算需求时,LLS将中断工作流1的作业的执行,从而延长了工作流1的平均响应时间。而默认调度器上作业的优先级顺序不会变化,作业一旦开始执行,则会一直执行直到结束,作业的执行不会被中断,从而使得具有更多空闲时间工作流中作业的平均响应时间低于LSS调度器。

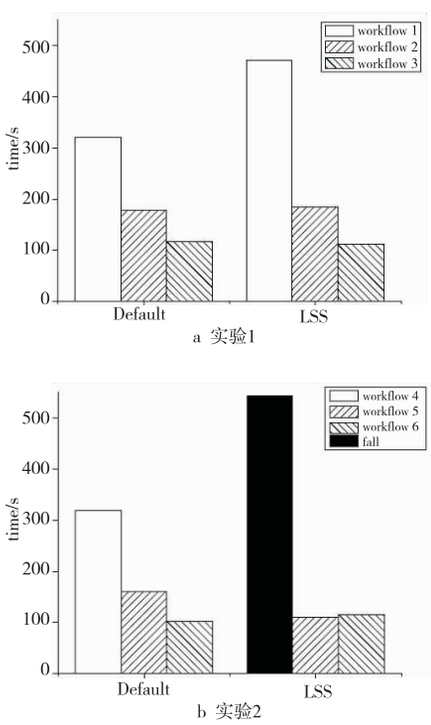

Figure 4　Average response time

图4　作业的平均响应时间

实验结果表明,不管是轻负载还是重负载,LSS调度器中作业的成功率均高于默认调度器,但是在作业的平均响应时间方面,时限要求较低的工作流中LSS调度器的效果会低于Hadoop的默认调度器。而硬实时调度器的首要设计目标是提高时限作业的成功执行率,因此在处理硬实时作业时,LSS调度器的性能得到了提升。

5 结束语

针对Hadoop现有调度器在处理硬实时作业上的不足,本文设计和实现了一种基于空闲时间的调度器LSS。为了提高集群在处理硬实时作业时的整体成功率,减轻集群的负载,LSS设计过程中采用了四个机制:(1)用空闲时间描述作业的优先级;(2)动态预测作业的空闲时间,并实时更新预测结果;(3)实时更新作业队列中作业的优先级顺序;(4)根据空闲时间的预测结果,提前结束不能成功执行的作业。实验结果表明,LSS能有效提高时限作业的成功执行率。预测空闲时间时,本文考虑的是同构集群,因此如何预测异构集群上的空闲时间将是本文的下一步研究工作。

参考文献:

[1] Dean J, Ghemawat S. MapReduce:Simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1):107-113.

[2] Lee K, Lee Y, Choi H, et al. Parallel data processing with MapReduce:A survey[J]. ACM SIGMOD Record, 2012, 40(4):11-20.

[3] Li Rui, Wang Bin. MapReduce in text processing[J]. Journal of Chinese Information Process, 2012,26(4):9-20. (in Chinese)

[4] Kc K, Anyanwu K. Scheduling Hadoop jobs to meet deadlines[C]//Proc of the 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010:388-392.

[5] Ma Xiao-yan, Hong Jue. A mutil-resource fair scheduler of Hadoop[J]. Journal of Integration Technology, 2012,1(3):66-71. (in Chinese)

[6] Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling:A simple technique for achieving locality and fairness in cluster scheduling[C]//Proc of the 5th European Conference on Computer Systems, 2010:265-278.

[7] Guo Z, Fox G. Improving MapReduce performance in heterogeneous network environments and resource utilization[C]//Proc of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012:714-716.

[8] Xu Cheng, Liu Hong, Tan Liang. New mechanism of monitoring on Hadoop cloud platform[J]. Computer Science, 2013,40(1):112-117. (in Chinese)

[9] Li Xin,Zhang Peng. Improvement and implementation of fair scheduling algorithms in Hadoop cluster[J]. Computer Knowledge and Technology, 2012(1):166-168. (in Chinese)

[10] Wei Xiao-hui, Fu Qing-wu, Li Hong-liang. Resource forecast delay algorithm for Hadoop systems[J]. Journal of Jilin University, 2013(1):101-106. (in Chinese)

[11] Ning Wen-yu, Wu Qing-bo, Tan Yu-song. MapReduce oriented self-adaptive delay scheduling algorithm[J]. Computer Engineering & Science, 2013,35(3):52-57. (in Chinese)

[12] Dong X, Wang Y, Liao H. Scheduling mixed real-time and non-real-time applications in MapReduce environment[C]// Proc of the 17th IEEE International Conference on Parallel and Distributed Systems, 2011:9-16.

[13] Lin Qing-ying. Cloud computing model on Hadoop[J]. Modern Computer, 2010(7):114-116. (in Chinese)

附中文参考文献:

[3] 李锐, 王斌. 文本处理中的MapReduce技术[J]. 中文信息学报, 2012,26(4):9-20.

[5] 马肖燕, 洪爵. 多资源公平调度器在Hadoop中的实现[J]. 集成技术, 2012,1(3):66-71.

[8] 许丞, 刘洪, 谭良. Hadoop云平台的一种新的任务调度和监控机制[J]. 计算机科学, 2013,40(1):112-117.

[9] 李鑫, 张鹏. Hadoop集群公平调度算法的改进与实现[J]. 电脑知识与技术, 2012(1):166-168.

[10] 魏晓辉, 付庆午, 李洪亮. Hadoop平台下基于资源预测的Delay调度算法[J]. 吉林大学学报:理学版, 2013(1):101-106.

[11] 宁文瑜, 吴庆波, 谭郁松. 面向MapReduce的自适应延迟调度算法[J]. 计算机工程与科学, 2013,35(3):52-57.

[13] 林清滢. 基于Hadoop的云计算模型[J]. 现代计算机, 2010(7):114-116.

作者简介:

杨浩(1990-),男,湖北公安人,研究生,研究方向为云计算和数据挖掘。**E-mail**:yanghaogn@163.com

YANG Hao, born in 1990, MS candidate, his research interests include cloud computing, and data mining.

滕飞(1984-),女,山东泰安人,博士,讲师,CCF会员(E200030587M),研究方向为云计算、调度和资源优化。**E-mail**:fteng@swjtu.edu.cn

TENG Fei, born in 1984, PhD, lecturer, CCF, member(E200030587M), her research interests include cloud computing, resource scheduling, and resource optimization.