一种Hadoop中基于作业类别和截止时间的调度算法

李 婴 滕 飞 李天瑞 杨 浩
(西南交通大学信息与科学技术学院 成都 610031)

摘 要 Hadoop是一种开源可靠的分布式计算框架,而MapReduce是处理超大规模数据集的编程模型。鉴于Hadoop内置的调度器不能很好地处理类别不同且有截止时间的作业的调度,提出了一种基于作业类别和截止时间的作业调度算法。作业分为CPU密集型和I/O密集型,并根据截止时间设置优先级来实现作业的调度。实验结果表明,该算法在充分利用集群的CPU和磁盘I/O的同时,能满足作业的截止期需求,当同一时间段内截止时间相近时算法达到最优,当某一队列中作业截止时间均比另一种队列短时,算法效率最低。

关键词 调度算法,截止时间,作业类别,MapReduce,Hadoop
中图法分类号 TP399 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.6.006

Scheduler Algorithm Based on Type Specific and Deadline in Hadoop

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

Abstract Hadoop develops open-source software for reliable, scalable, distributed computing. MapReduce is a programming model and an associated implementation for processing large data sets. Because the built-in Hadoop scheduler cannot handle the different type and deadline based jobs, we proposed a scheduler algorithm based on type specific and deadline. We specified these jobs into CPU-bound and I/O-bound and gave priority to jobs according to the deadline. The results of experiments show that the proposed algorithm not only makes full use of the cluster's CPU and I/O resource, but also meets the jobs' deadline. If the deadline is almost the same at a period of time, the algorithm is the best. But if jobs' deadlines from one queue are all shorter than another queue, the efficiency of the algorithm achieves the minimum.

Keywords Scheduler algorithm, Deadline, Job-type, MapReduce, Hadoop

1 引言

信息时代中机器学习、科学分析、天体物理和生物信息学等很多领域需要存储、分析和处理日益膨胀的数据,传统单机环境并不能满足这种需求,云计算应运而生。其中Hadoop是开源可靠的分布式计算框架,它具有成本低、可靠性强、效率高、可伸缩性等优点[1],但在作业调度等其他方面仍存在一些不足:它所提供的调度算法不能很好地满足用户不同的个性化需求。

当前主要有中央式调度、双层调度和共享式调度3类集群管理系统[2]。中央式调度的特点是资源的调度和作业的管理功能在一个进程中完成,典型例子是Hadoop中的JobTracker。这种调度的优点是集群规模较小时能大大缩短作业的响应时间。双层调度简化了中央调度,将调度策略放入各个应用程序。共享状态调度将上述两种集中式资源调度模块简化成持久化的共享数据,共享数据即集群实时资源信息。本文拟使用小规模集群研究中央式调度中Hadoop调度算法。

调度是对作业的执行顺序和计算资源的分配进行控制。很多学者对Hadoop调度算法进行研究并提出了不同的调度算法。Hadoop调度算法研究主要可以分为两大方向:一种是针对MapReduce架构的调度算法研究,主要是通过减少MapReduce架构中内部数据迁移、I/O开销、时间预期等实现优化;另一种是针对Hadoop的公平调度算法研究,主要通过对作业的合理调度实现对算法的优化[3]。

目前Hadoop内置的调度器是先进先出调度器,这是一种单用户、单队列的调度算法。为了克服这种缺点,Yahoo!和Facebook分别提供了容量调度器和公平调度器这两种多用户多队列的调度算法[1]。文献[4]提出了使用先进先出的调度策略预估作业类型,但这种策略不能很好地处理作业优先级。文献[5,6]研究了基于截止时间的算法,但是并没有考虑作业类别。在集群中,大量不同类别的作业被提交,我们从对CPU和I/O的不同使用情况出发,通过区分不同作业的类别,大大提高集群硬件资源利用率。

本文结合作业类别和截止时间,从满足区分作业类型和截止时间需求从而提高集群资源利用率的角度出发,提出了

到稿日期:2014-04-12 返修日期:2014-05-13 本文受国家自然科学基金(61202043,61175047)资助。
李 婴(1988—),女,硕士生,主要研究方向为云计算、数据挖掘等,E-mail:lizhao0722@126.com;滕 飞(1984—),女,讲师,主要研究方向为云计算、调度、资源优化等,E-mail:fteng@home.swjtu.edu.cn;李天瑞(1969—),男,教授,博士生导师,主要研究方向为智能信息处理、粗糙集、粒计算、云计算等,E-mail:trli@swjtu.edu.cn;杨 浩(1990—),男,硕士生,主要研究方向为云计算、数据挖掘等。

基于作业类别和截止时间的调度算法。该算法分为两部分：

1)作业类别的区分。根据 MapReduce 的执行过程，将数据处理分为 5 个阶段，根据不同阶段数据的吞吐量和磁盘 I/O 的比值，将作业区分为 I/O 密集型和 CPU 密集型，并将其放入两种不同的队列中。

2)基于截止时间的调度。作业放入两种不同的队列后，根据截止时间的要求，为作业设置不同的优先级，截止时间相对当前时间越近的作业优先级越高。

2 相关研究

调度算法的重要性使得众多专家和学者投身其中并提出了很多算法。对 MapReduce 调度算法的研究主要集中在数据本地性、实时性、异构集群和预估任务的完成时间等方面。

在调度算法研究方面，田超等研究了区分作业类别的调度算法，其将提交的作业划分为 3 种类别，但仍然使用先进先出调度，并未给作业设置优先级[4]。滕飞等提出了基于截止时间的硬实时任务调度算法 SPS，其能最大限度保证作业在截止时间内完成，大大提高了作业在截止时间内完成的几率[5]。Kc 等也指出了一种将截止时间作为用户输入一部分的调度算法[8]。张晓宏等提出了 NKS 算法以提高任务执行时的数据本地性特征，优先选择数据在本机的作业，通过计算数据不在本机的可能性，选择最高可能性的作业进行调度执行，但该算法不适用于同构集群[7]。在基于截止时间调度算法研究方面，唐卓等提出了扩展的截止时间算法 MTSD，其允许用户指定作业完成时间，并确保作业尽量在截止时间内完成[8]。宁文瑜等设计并提出了自适应延迟调度算法，其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[9]。杨浩等设计并实现了一种基于空闲时间的硬实时调度器 LSS，即在调度过程中，动态估算作业的空闲时间，并实时更新作业队列中作业的优先级[10]。

当前研究主要集中在数据本地性、任务执行时间预估、作业实时性及作业类别等方面，但区分作业类别后在各自的队列中仍沿用系统内置的算法[4]，尽管有些学者对截止时间进行研究，但未涉及作业类别[5,8]。本文结合作业类别和截止时间，提出一种基于作业类别和截止时间的调度算法。

3 一种基于作业类别和截止时间的调度算法

3.1 Hadoop 简介

Hadoop 主要由 3 部分组成：HDFS、MapReduce 和 Hadoop Common[1]。HDFS 部署在低成本的硬件设备上，具有很好的容错性。MapReduce 是一个并行编程模型，具有并行计算、数据容错、数据冗余等特性，没有并行开发经验的用户也能很快掌握此模型。MapReduce 包含一个 Jobtracker 和多个 Tasktracker。Jobtracker 接受用户提交的任务并将任务分发到 Tasktracker 上，Tasktracker 则是对数据进行分析计算的节点。Tasktracker 之间以及 Tasktracker 和 Jobtracker 之间通过心跳机制通信。MapReduce 有 Map 和 Reduce 两个阶段。数据在 Map 阶段被分成默认 64M 的数据块，并生成键值对，这些键值对在 Reduce 阶段通过键值重新分组。Reduce 分为复制 Map 结果到 Reduce 节点、对结果进行排序和计算 3 个阶段。

3.2 Hadoop 数据处理流程

在执行 Map 任务时，首先读取 HDFS 上的块数据，并形成键值对；随后根据 Partition 接口决定将每个键值对交由哪个 Reduce 进行处理，默认是对 Key 进行哈希取值再对 Reduce 数量取模。Map 的结果会首先置于内存缓冲区，当缓冲区满时启动溢写线程将数据溢写到磁盘。溢写线程启动后会对 Key 进行排序。所有溢写完成后会对溢出文件做归并操作(Merge)，即将多个文件归并到一起，Map 阶段任务结束。

Reduce 进程启动复制线程，获取 Map 任务的输出文件，并对文件不断进行归并，最终生成一个输出文件。

本文将整个过程分为 5 个阶段，即初始化数据(Map Init Data，MID)、Map 处理数据(Map Compute Data，MCD)、将 Map 端数据存到磁盘(Map Output Data，MOD)、Shuffle Map 结果(Shuffle Data Output，SOD)、Reduce 端输入数据(Reduce Data Input，RDI)，如图 1 所示。

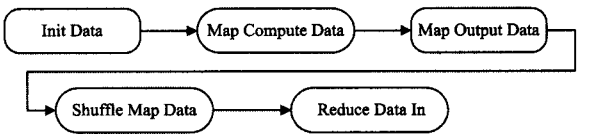

图 1 Mapreduce 数据处理过程

3.3 作业类别的区分

用户将 CPU 密集型和 I/O 密集型作业提交到集群，这些作业在同一时刻会出现抢占相同资源的情况。通过区分两种类别的作业，交替地提交作业，可实现同一时间段内两种类别作业对不同资源的有效使用。

根据 MapReduce 任务执行过程中 CPU 和 I/O 的利用情况，根据需要将数据处理过程分为上述 5 个阶段。在此过程中假设每个 Map 所处理的任务量相同。设 α 是单个 Map 输入数据量和输出数据量的比例因子，作业类别不同，比例因子也不同[4]。则有：

$$MOD=MID*\alpha \tag{1}$$

假设每个相同作业的 α 因子相同，这是因为 Map 过程中的输出结果就是 Shuffle 阶段的输出，所以 SOD 和 MOD 相同。则有

$$SOD=MOD \tag{2}$$

RDI 则不同于 SOD，因为在 Map 阶段 Shuffle 的数据是每个溢出文件，而 Reduce 阶段中的 RDI 是一个或多个溢出文件合并的数据文件。所以 RDI 与正在运行的 Reduce 数量(Running Reduce，RR)、总 Reduce 任务数量(Total Reduce，TR)和节点个数有关，这里我们假设节点个数为 k。则有：

$$RDI=\frac{RR}{TR}*SOD*k \tag{3}$$

在 MapReduce 的 5 个阶段中，不同作业会不同程度地占用 I/O 带宽和 CPU 资源，所以考虑将这几个阶段资源利用率之和与磁盘 I/O 利用率相比(Disk I/O Rate，DIOR)。

如式(4)所示，不同的作业共享同一 I/O 资源，若此阶段资源利用率之和(MOD+MID+SID+RDI)小于磁盘 I/O 利用率，则认为此任务为 CPU 密集型任务。以下设每个节点上 Map 任务的个数为 m，Map 任务总的完成时间为 MT，则有：

$$\frac{(MID+MOD+SOD+SID)*m}{MT}=\frac{((1+2\alpha)*MID+SID)*m}{MT}<DIOR \tag{4}$$

满足式(5)所示的作业被定义为 I/O 密集型任务。多个此类任务被提交给集群后，大量占用 I/O 资源，使得 I/O 阻塞，进而影响任务的持续推进。

$$\frac{(MID+MOD+SOD+SID)*m}{MT}=$$

$$\frac{((1+2\alpha)*MID+SID)*m}{MT}\geqslant DIOR \quad (5)$$

3.4 基于截止时间的算法

针对作业截止时间的要求，在区分了作业的类别并将其分配到不同的作业队列中后，可为作业设置不同优先级，使作业在截止时间内完成。对很多应用来说，在截止时间内完成是非常重要的。例如，谷歌使用 MapReduce 模型构建了基于云平台的倒排索引，该索引需要及时完成更新[11]；网页个性化服务中也需要根据客户的设置和搜索历史及时分析用户的意图并提供其所需要的信息，这就需要在指定的时间期限内完成[12]。类似的应用在规定的时间内完成至关重要。

1)截止期算法的调度策略。通过预测模型区分了作业类别后，这些作业会被分至 CPU 队列和 I/O 队列两个不同的队列中。这两个队列中分别维持着一个优先级序列。根据用户的截止时间要求，越接近最后完成期限的作业优先级越高，Jobtracker 会选择优先级最高的作业执行。

设 $T_{deadline}$ 为作业完成的最后期限，T_{now} 为作业提交时系统的当前时间，T_{define} 为用户设置的作业从提交到完成所能容忍的最长时间。

$$T_{deadline}=T_{now}+T_{define} \quad (6)$$

根据式(6)，T_{define} 越小，则作业优先级越高。

2)抢占式调度。调度分为可抢占式调度和非抢占式调度。本文采用抢占式调度，即在作业运行过程中，将空闲资源预留给更高优先级的作业，而当前作业将会等待，直到有资源被释放。

3)动态优先级顺序。在运行过程中，作业间断式提交，已提交的作业已有固定的优先级顺序，一旦提交新的作业，自动调整其在队列中的优先级。

4)截止时间已到的作业。设定作业的优先级后，由于资源有限，存在截止时间已到但还未完成或还未开始的作业，此时终止此类作业的运行，将资源留给其他作业，以最大限度保证后续作业能在截止时间内完成。

3.5 基于作业类别和截止时间的调度算法

本文将整个调度算法分为两个部分，在作业提交之后首先区分作业为 CPU 密集型或 I/O 密集型，并将其划分到两种不同的队列中，这两种队列维持一个优先级顺序，Jobtracker 根据队列的优先级顺序选择作业运行。算法过程如下(如图 2 所示)：

1)运行 Hadoop 自带测试用例 TestDFSIO，获得集群 DIOR；

2)用户提交作业到客户端，当有空闲槽时启动一个 Map 任务到空闲槽运行，根据历史运行信息，得出 MID、MOD、MT 等信息；

3)比较式(4)、式(5)和 DIOR，将两种作业划分到不同的作业队列中；

4)两种队列分别根据用户设置的截止时间设置作业优先级，并动态调整优先级；

5)当有空闲槽时，两个队列中 Deadline 最早的作业先启动(如优先启动为 CPU 队列中作业)；

6)启动 I/O 队列里优先级最高的作业；

7)如此循环交替运行两种队列中的作业。

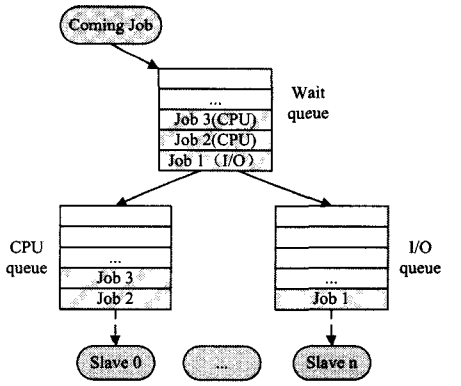

图 2 调度策略

基于作业类别和截止期的调度算法的伪代码如下所示。

算法 SortJobs()

输入：Job，表示作业；CPUQueue，表示 CPU 队列；I/OQueue，表示 I/O 队列

输出：Job

开始：

/ * Divide the job type based on the formula and set priority to the job. On each queue based on the user-defined deadline * /

1. IF freeslot>0
2. Monitor mapToRun(map)
3. ELSE
4. Return
5. / * Divide job type according to task history * /
6. IF $\frac{(MID+MOD+SOD+SID)*m}{MT}=\frac{((1+2\alpha)*MID+SID)*m}{MT}<$ DIOR
   Put CPUQueue(Jobs)
7. ELSE
8. Put I/Oqueue(Jobs)
9. / * Set priority on each queue * /
10. $T^{job_1}_{deadline}$=getDeadline(Job_1)
11. $T^{job_2}_{deadline}$=getDeadline(Job_2)
12. IF $T^{job_1}_{deadline}<T^{job_2}_{deadline}$
13. $P(Job_1)>P(Job_2)$
14. ELSE
15. $P(Job_1)<P(Job_2)$
16. IF ComesNewJob
17. Reset P(Jobs)

结束

4 基于作业类别和截止时间的调度算法实验

4.1 实验方案

用 6 台 DELL PowerEdge R710 (Xeon E5506/12G/600G)服务器搭建了 Hadoop 云计算实验平台，其中 1 个主节点，5 个从节点，每个从节点包括 2 个 Map 插槽和 2 个 Reduce 插槽，主节点与从节点通过千兆以太网相连接。所用的 Linux 内核版本为 2.6.18-164.el5，Java 版本为 1.7.0_01，Hadoop 版本为 1.0.3。

由于每个从节点包括 2 个 Map 槽和 2 个 Reduce 槽，因此式(4)和式(5)中 m 的值为 2。使用 Hadoop 中的 TestDFSIO 测试用例获得集群的磁盘 I/O 利用率，得到本集群的 DIOR 为 14.6MB/s。使用两种不同的作业流作为测试用例。Terasort 是 Hadoop 中的一个排序作业，它的 MID 为 64M，

MOD为64M，MT为8s，根据式(5)，它属于I/O类作业。Grep程序实现的是对指定文档中指定单词的词频进行计算，它的MID为64M，MOD为0.49M，MT为13s，根据式(4)，它属于CPU类型作业。

实时系统中常用作业流来描述提交的作业，同一作业流中提交同一类型的作业，作业流用 $W_i=(\lambda_i,D_i,S_i,M_i,R_i)$ 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，若 $\lambda_i=1/40$，则该作业流中作业的到达时刻分别为39、80、118…；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 表示作业包括的Map任务的数目；R_i 表示作业包括的Reduce任务的数目。

如表1所列，本实验有TeraSort和Grep两种作业，每个作业有3个作业流，每种作业流有5个作业。因为实验结果依赖于作业的截止时间，所以进行了3次实验，以获得不同情况下作业的运行情况，如表2所列。

表1 作业类型

程序	MID(M)	MOD(M)	MT(s)
Terasort	64	64	8
Grep	64	0.49	13

表2 作业流

参数	负载一	负载二	负载三
Terasort	W1=(1/60,60,1.2G,20,2)	W1=(1/75,75,1.2G,20,2)	W1=(1/110,110,1.2G,20,2)
	W2=(1/50,50,0.6G,10,2)	W2=(1/60,60,0.6G,10,2)	W2=(1/100,100,0.6G,10,2)
	W3=(1/40,40,0.3G,5,2)	W3=(1/45,45,0.3G,5,2)	W3=(1/80,80,0.3G,5,2)
Grep	W4=(1/60,60,1.2G,20,2)	W4=(1/75,75,1.2G,20,2)	W4=(1/70,70,1.2G,20,2)
	W5=(1/50,50,0.6G,10,2)	W5=(1/60,60,0.6G,10,2)	W5=(1/60,60,0.6G,10,2)
	W6=(1/40,40,0.3G,5,2)	W6=(1/45,45,0.3G,5,2)	W6=(1/50,50,0.3G,5,2)
调度器	FIFO,SPS,TSD		

4.2 实验结果和结果分析

本文有2种类别作业，即CPU密集型和I/O密集型作业，作业按照泊松分布的方式到达。每种作业有3个作业流，每个作业流5个作业，共30个作业。

SPS调度算法[5]仅考虑作业的截止期，而没有区分作业的类型。TSD调度算法与SPS进行比较，其成功率(在截止时间到来之前已经完成作业占总作业的比值)如图3所示。

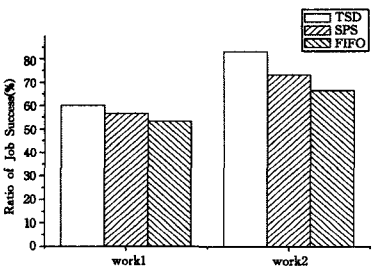

图3 负载一和负载二作业成功率

从图3可以看出，提交30个作业后，SPS调度算法的作业成功率明显高于FIFO调度算法，这是因为SPS算法考虑作业的截止时间，给截止时间最近的作业设置最高的优先级，并将截止时间到来后还未完成的作业剔除，将资源预留给后面的作业。负载二情况下，SPS调度算法成功执行了22个作业，而TSD调度算法成功执行了25个作业，比未区分作业类型的SPS算法高，这是因为TSD算法交替提交任务，可以让不同作业类型使用不同的资源，以实现对集群资源的充分利用。

统计负载二情况下的作业完成时间(作业完成时间为一个负载从提交到完成所用全部时间，负载一与负载二情况相同)，如图4所示。可以看出，在负载二情况下，Hadoop自带调度算法FIFO作业完成时间为351s，SPS调度算法完成时间为369s，TSD算法为356s。FIFO略快于SPS，这是因为SPS设置了优先级，当高优先级作业到来后会分配到空闲资源，当前作业则会等待，这样会延长总的任务执行时间。同理，FIFO也略快于TSD，但是其因为能充分利用不同类别的资源，所以比SPS更快一些。

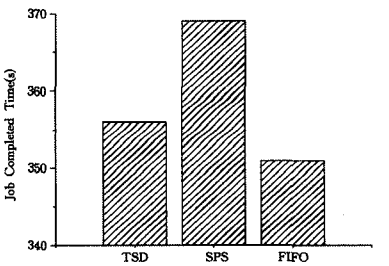

图4 负载二作业完成时间

统计两种负载状况下每个作业流中作业的平均响应时间(作业的响应时间为单个作业从开始提交到完成所需的时间)，如图5和图6所示。可以看出，TSD调度算法在响应速度上比SPS要快，在数据量越大的情况下，差距越明显。一般情况下，由于实验中设置的负载二为轻负载，因此负载二的作业流响应时间会比负载一长，但是从图5和图6可以看出，负载二的作业流3、4、5的响应时间要比负载一短，出现这种现象的原因有两种：第一，不同的负载被剔除作业的个数不同，其响应时间不同；第二，资源在分配时，优先级不同，若负载二中作业流3中某个作业优先级低，则其资源会被其他作业抢占，它的整个作业流响应时间也会变长。

由表2可以看出，我们设计的负载三的作业流4、5和6的截止时间均比作业流1、2和3短。如图7所示，分别配置了TSD和SPS调度算法，得出SPS的作业成功率为66%，而TSD的作业成功率为56%。这是因为Grep类型作业的截止时间设置得都比TeraSort短，在类似这种情况下，TSD调度算法的效率最低。由此我们可以得出，在一段时间内，当两种类别的作业截止时间相差较小时，算法效率较高；而当出现类似上述极端情况时，算法效率最低。

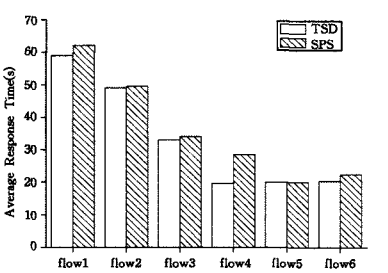

图5 负载一作业平均响应时间

(下转第45页)

semble by voting [C]// Proceeding of the International Conference on Information and Communication System. Amman, Jordan, 2009: 1-5

[12] Yang Yan, Wang Hong-jun, Lin Chao, et al. Semi-supervised Clustering Ensemble Based on Multi-ant Colonies Algorithm [C]// Rough Sets and Knowledge Technology 7th International Conference. Chengdu, China, 2012: 302-309

[13] Wagstaff K, Cardie C, Rogers S. Constrained k-means clustering with background knowledge [C]// Proceedings of the 18th International Conference on Machine Learning. San Francisco, CA, USA, 2001: 577-584

[14] Klein D, Kamvar S D, Manning C. From instance-level constraints to space-level constraints: marking the most of prior knowledge in data clustering [C]// Proceedings of the 19th International Conference on Machine Learning. San Francisco, CA, USA, 2002: 307-314

[15] Pedrycz W, Waletzky J. Fuzzy Clustering with Partial Supervision [J]. IEEE Transactions on Systems, Man, and Cybernetics, 1997, 27(5): 787-795

[16] Karypis G, Kumar V. Multilevel K-Way partitioning scheme for irregular graphs [J]. Journal of Parallel Distributed Computing, 1998, 41(2): 278-300

[17] Ng A, Jordan M, Weiss Y. On Spectral Clustering: Analysis and an Algorithm[C]// Advances in Neural Information Processing Systems. 2001, 14: 849-856

[18] Blake C L, Merz C J. UCI repository of machine learning databases [EB/OL]. 2012-05-01[2012-12-01]. http://archive.ics.uci.edu/ml

[19] Modha D, Spangler W S. Feature weighting in k-means clustering [J]. Machine Learning, 2003, 52(3): 217-237

(上接第 31 页)

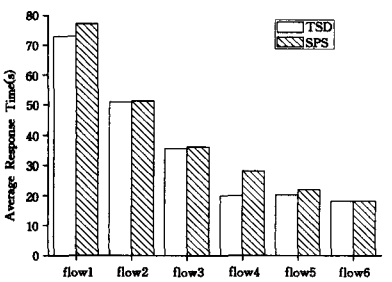

图 6 负载二作业平均响应时间

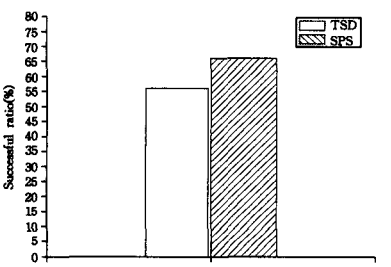

图 7 负载三作业成功率

结束语 针对不同作业类别，本文从提高集群资源利用率的角度出发，设计了一种基于作业类别和截止时间的调度算法。该调度算法包括两部分：1)将作业划分为 CPU 密集型作业和 I/O 密集型作业，并将其置于两种不同的队列中；2)根据作业执行截止时间，为作业设置不同的优先级，截止期限越近的作业优先级越高。该算法充分利用了集群的 CPU 和 I/O 资源，优于 Hadoop 默认调度算法。后续研究的重点在于对类型动态变化的作业调度的研究，如某些类型的作业可能早期使用 CPU 较多但后期则使用 I/O 较多，此种类型作业的调度尚有待进一步深入研究。

参 考 文 献

[1] White T. Hadoop 权威指南[M]. 周敏，译. 北京：清华大学出版社，2011：23-55
White T. Hadoop: The Definitive Guide[M]. Zhou Min. Beijing: Tsinghua University Press, 2011: 23-55

[2] Schwarakopf M, Konwinski A. Omega: flexible, scalable schedulers for large compute clusters[J]. EuroSys'13 Proceeding of the 8th ACM European Conference on Computer Systems. 2013: 351-364

[3] 范帆. Hadoop 中基于优先级的调度算法研究[D]. 上海：复旦大学，2012，8
Fan Fan. A Priority-basd Scheduling Algorithm for Hadoop[D]. Shanghai: Fudan Unversity, 2012, 8

[4] Tian Chao, Zhou Hao-jie, He Yong-qiang, et al. A Dynamic MapReduce Scheduler for Heterogeneous Workloads[C]// Eigth International Conference on Grid and Cooperative Computing (GCC '09). 2009: 218-224

[5] Teng Fei, Yang Hao, Li Tian-rui, et al. Scheduling real-time workflow on MapReduce-based cloud[C]// 2013 Third International Conference on Innovative Computing Technology. 2013: 117-122

[6] Kc K, Anyanwu K. Scheduling Hadoop Jobs to Meet Deadlines [C]// 2010 IEEE Second International Coference on Cloud Computing Technology and Science. 2010: 388-392

[7] Zhang Xiao-hong, Ju Shuai, Jiao Zhi-bin. A Scheduling Method Based on Deadlines in MapReduce[J]. Electrical, Information Engineering and Mechatronics 2011 Lecture Notes in Electrical Engineering, 2012, 138: 1585-1592

[8] Tang Zhuo, Zhou Jun-qing, Li Ken-li, et al. MTSD: A task scheduling algorithm for MapReduce base on deadline constraints[C]// 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. 2012: 2012-2018

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

[10] 杨浩，滕飞，李天瑞，等. Hadoop 平台中空闲时间调度器的设计与实现[J]. 计算机工程与科学，2013(10)：125-131
Yang Hao, Teng Fei, Li Tian-rui, et al. Design and implementation of a least spare time scheduler for Hadoop[J]. Computer Engineering & Science, 2013(10): 125-131

[11] 陈国营. 基于 MapReduce 模型文本分类算法的研究[D]. 辽宁：辽宁大学，2013：1-10
Chen Guo-ying. Design and Implementation of Text Classification Algorithm Based on Hadoop[D]. Liaoning: Liaoning University, 2013: 1-10

[12] 韩定一. 云推荐—大数据时代的个性化互联网服务解决之道[J]. 程序员，2013(3)：16-17
Han Ding-yi. Cloud recommend—The road to solve personalized service on the era of big data[J]. Programmer, 2013(3): 16-17