基于优先级的Three-Queue调度算法研究

顾 宇 周 良 丁秋林
(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 针对Hadoop平台上调度算法存在的不足,提出了一种改进的调度算法——Triple-Queue算法。在充分考虑数据的本地性后,Triple-Queue算法设计了一种改进的优先级计算模型,以有效地区分用户作业的等级,同时又保证一定程度的公平性,进而减小作业执行时间,避免系统资源浪费。实验结果表明,随着数据量的提高,该算法执行效率明显提高,同时能够较好地解决数据本地性问题。

关键词 调度,Triple-Queue,数据本地性,MapReduce
中图法分类号 TP393 **文献标识码** A

Research of Three-Queue Scheduling Algorithms Based on Priority

GU Yu ZHOU Liang DING Qiu-lin
(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract For solving the shortage of scheduling algorithms on the Hadoop platform, the paper proposed an improved scheduling algorithm-Triple-Queue algorithm. After taking full account of data locality, the Triple-Queue algorithm designs a improved computational model of priority, which can distinguish the user's job levels clearly and ensure a certain degree of fairness, so as to reduce the job execution time and avoid wasting system resources. The results of experiment show that the algorithm improves the efficiency significantly and solves the problem of data locality better with the increased amount of data.

Keywords Scheduling, Triple-Queue, Data locality, Map reduce

1 引言

近年来随着计算机应用的不断拓展和深入,许多计算机应用领域需要处理的数据规模越来越大,数据量也从GB级、TB级向PB级发展,这导致了云计算和数据密集型运算领域迅速发展,Hadoop就这样应运而生了。Hadoop是Apache软件基金会组织下的一个开源云计算平台,可以在大量廉价的硬件设备组成的集群上运行分布式应用程序。MapReduce是Hadoop平台上的核心部分,作为一个分布式计算模型,旨在处理大规模数据并生成大量的数据结果输出[1]。

MapReduce编程模型将海量数据处理操作抽象为映射Map和规约Reduce两种操作,从而大大降低了分布式并行计算程序的编写难度。Map操作就是将数据打散,即将输入的数据文件的键值对转换成中间键值对。Reduce操作,则是实现数据聚集,即将Map阶段输出的中间键值对转换成最终的结果输出。MapReduce框架包含一个主节点(Master)或称为Jobtracker和若干工作节点(Worker)或称为Tasktracker。其中,Jobtracker将用户提交的任务划分成Map任务和Reduce任务,然后按照调度算法将其派发至各个Tasktracker上。因此,Jobtracker是用户和MapReduce框架之间的交互点,也是执行调度的核心部分。

在MapReduce调度中,数据的本地性是调度算法重点考虑因素,其基本要求是应尽可能地把计算任务放在距离所需数据较近的工作节点上执行,从而减少因数据传输造成的网络带宽资源浪费,进而提高集群的吞吐率,节省带宽资源,增强系统运行效率。据此,本文结合MapReduce调度算法的研究现状,重点探讨MapReduce框架中数据的本地性问题。在此基础上,提出了一种MapReduce框架下的Triple-Queue调度算法,设计了一种改进的优先级计算模型,并给出了Triple-Queue调度算法的实现。

2 研究现状

在分布式集群系统中,合理的调度算法对提高大规模数据处理的效率至关重要。国内外许多学者对MapReduce调度算法都进行深入的研究,先后提出了很多的调度算法[2-9]。文献[2]提出了一种公平调度算法,主要针对在集群上运行的数据密集型作业和交互式作业,尽可能地让资源平台到每个作业调度。文献[3]中提出,当某个作业由于不能得到相应的资源保证而无法执行时,将此作业适当的等待,等到获得足够的资源后再执行。文献[4,7]是通过计算作业的预计执行完成时间,来动态地分配系统资源数,适合实时作业系统,但是面对混合作业系统有很大的局限性。文献[5]中的调度算法更多地关注作业公平性,按照一定的比例对系统资源进行分配共享。文献[6]中提出了一种调度,根据作业执行的时间限

顾 宇(1987—),男,硕士生,主要研究方向为海量数据处理,E-mail:guyu_111@163.com;周 良(1966—),男,副教授,硕士生导师,主要研究方向为信息系统集成与知识工程;丁秋林(1936—),男,教授,博士生导师,主要研究方向为企业信息化和信息系统集成。

制作业的调度，借此提高系统的资源利用率。文献[8]中根据系统历史信息动态调整 Map 和 Reduce 任务各阶段的时间比例，以找出真正需要优先执行的任务。文献[9]中通过计算作业的响应时间即作业预计完成结束时间与作业提交时间的差值大小来确定执行作业的优先级顺序。

以上文献中提出的这些 MapReduce 调度算法，在某些特定领域具有优势，但是如遇到以下的任务调度场景时（见表1），仍存在一些问题。

表1 任务调度场景

工作节点	存储的数据	本地执行任务
W1	D1	T1
W2	D2	T2
W3	D3,D4	T3,T4
W4		

假设工作节点以 W4—W3—W2—W1 的顺序申请任务时（每次只能申请一个任务），按照以上提出的那些调度算法[2-9]，满足数据的本地性要求，得到的任务列表为 D1—D2—D3—D4。这个结果数据需要传输 3 次，但是理想的结果应为 D4—D1—D2—D3 或者 D3—D1—D2—D4，数据只需要传输 1 次。可见以上的调度算法在判定执行任务优先级时均没有真正意义上的数据本地性，本文提出的 Triple-Queue 调度算法，考虑作业优先级（JobPriority）、任务优先级（TaskPriority）和工作节点优先级（WorkerPriority）3 个方面，提出了新的优先级计算模型，能够较好地解决了数据本地性问题。

3 Triple-Queue 调度算法

3.1 优先级判定策略

在已有的调度算法中，我们判定优先级（Priority）高低总是从作业优先级（JobPriority）参数角度着手，但是仅仅根据 JobPriority 来决定作业的执行调度是不够精确的，因为没有考虑数据本地性问题，一旦考虑到数据的本地性问题，优先级的判定将变得更加复杂。因此，在原有作业优先级参数的基础上，又引入了两个参数：任务优先级（TaskPriority）和工作节点优先级（WorkerPriority），融合 3 个参数建立了一个新的优先级计算模型，下面我们将详细叙述这 3 个参数模型的建立。

（1）作业优先级（JobPriority）计算模型

为了计算 JobPriority，我们定义几个常量。JobSize 表示提交作业的大小，taskNum 表示完成作业需要的任务数，averageTaskSize表示每个任务平均执行长度，priorityValue 表示作业的权值，weighValue 是表示根据作业权值大小得到可以获得的任务数。所以，某种程度上 JobPriority 即是 weighValue 的大小。weighValue 推理过程如下：

①求出作业中每个任务平均执行长度：

$$averageTaskSize = JobSize / taskNum$$

②求出所有作业一次调度中，作业处理的数据总量 S：

$$S = \sum_{i=1}^{n} S[i] = \sum_{i=1}^{n} averageTaskSize[i] * weightValue[i]$$

③通常情况下，执行 map 任务的任务数会比执行 reduce 任务的任务数多得多，所以为了简单起见，用系统能够同时运行的 Map 任务的数目来表示系统所能处理 Task 的数目，记为：maptaskSum。因为 Map 任务的输入数据是输入数据的分块，大小即为 HDFS 中数据块的大小，记为 blocksize，所以每一次作业调度的数据量为：

$$S = \mathrm{Max}(num * jobnum * blocksize, maptaskSum * blocksize)$$

这里 *jobnum* 为一次作业调度中执行的作业数，*num* 是每个作业平均执行的任务数（假设每个作业一次只执行一个数据块）。

④作业 *job*[*i*] 对应的 *S*[*i*] 值与总 *S* 的比值等于对应的 *priorityValue*[*i*] 值与总 *priorityValue* 的比值：

$$\frac{averageTaskSize[i] * weightValue[i]}{S} = \frac{priorityValue[i]}{\sum_{i=1}^{n} priorityValue[i]}$$

⑤求出 *weightValue*[*i*] 的值大小：

$$weightValue[i] = \frac{S * priority[i]}{averageTaskSize[i] * \sum_{i=1}^{n} priority[i]}$$

（2）任务优先级（TaskPriority）计算模型

因为执行 Map 任务的任务数会比执行 Reduce 任务的任务数多得多，所以这里我们讨论的任务优先级主要是针对 Map 任务的。主节点 Master 需要为每一个 Map 任务计算调度优先级，其大小设置为 *TaskPriority*，*TaskPriority* 值越高的 Map 任务将更优先被调度。*TaskPriority* 的计算模型如下：

$$TaskPriority = \alpha T_1(LocalWorkerNum) + \beta T_2(WorkerTaskListLength) + \gamma T_3(TaskWaitTime) + C_1$$

式中，*LocalWorkerNum* 表示能够本地执行 Map 任务的工作节点数目，其值越大，*TaskPriority* 越小；

WokerTaskListLength 表示通过 Map 任务执行的工作节点的本地任务列表长度，其值越大，*TaskPriority* 越大；

TaskWaitTime 表示任务从执行初始化开始到目前时刻的时间，其值越大，*TaskPriority* 越大；

*C*1 表示应当考虑的其他影响因素。

这里定义 $T_1(LocalWorkerNum)$ 为单调递减函数，$T_2(WokerTaskListLength)$ 和 $T_3(TaskWaitTime)$ 为单调递增函数。

（3）工作节点优先级（WorkerPriority）计算模型

这里先举例子说明一下，假如工作节点 *A* 主要用来处理短作业，如实时查询，实时显示等；工作节点 *B* 主要是用来处理长作业的，如历史数据分析，数据挖掘等；当工作节点 *A* 和 *B* 同时申请任务时，显然工作节点 *A* 优先于 *B* 执行任务，所以得出 *WorkerPriority* 值越高，其上的任务越优先被执行。以上只是粗略描述了一下，*WorkerPriority* 要考虑的因素很多，其计算模型如下：

$$WorkerPriority = \begin{cases} MaxValue \\ \alpha W_1(RequestNum) + \\ \beta W_2(HistorySuccessRate) + C_2 \end{cases}$$

式中，*RequestNum* 表示工作节点最后一次获得执行任务之后申请任务的数目，其值越大，表示此工作节点负载越低，处理任务能力越强，因此有空闲任务时，此工作节点会优先调度执行。

HistorySuccessRate 表示工作节点历史执行任务成功率，其值越大，表示此工作节点处理任务能力越强，所以会优先执

行空闲任务。

$C2$ 表示应当考虑的其他影响因素。

当工作节点中本地执行任务列表不为空时，$WorkerPriority$ 可以被设为最大值 $MaxValue$，此工作节点被优先执行，否则按照计算模型进行计算。$W_1(RequestNum)$ 和 $W_2(HistorySuccessRate)$ 为单调递增函数。

综上 3 个优先级参数分析，得出最终的优先级判定计算模型，如下所示：

$$Priority=\alpha * JobPriority+\beta * TaskPriority+\gamma * WorkerPriority$$

对作业优先级、任务优先级和工作节点优先级 3 者进行加权求和，得出优先级的数值。在本文提出的 Triple-Queue 调度算法中，根据 Priority 值的大小，进行作业调度执行的决策，以此获得更好的系统执行效率。

3.2 算法描述

Triple-Queue 调度算法主要定义了 3 个队列：动态队列 DynamicQueue[]、备份队列 BackupQueue[]和就绪队列 ReadyQueue[]，其中 DynamicQueue[]是维护系统正在执行的作业；BackupQueue[]是维护因负载水平调整从 DynamicQueue[]中去除的正在执行的作业；ReadyQueue[]是维护用户提交的作业。Triple-Queue 调度算法流程描述如下：

(1)把用户提交的作业添加至 ReadyQueue[]队列中，根据 3.1 节提出的优先级计算模型，分别求出作业优先级、任务优先级和工作节点优先级，然后计算每个用户作业的优先级，最后从高到低进行排序。

(2)检测到当前系统中 Tasktracker 节点空闲时，Tasktracker 节点向 JobTracker 节点发出请求执行作业任务的消息。

(3)JobTracker 节点判断 Tasktracker 节点是否为慢节点，如果是慢节点，放弃执行后备任务，改为从 ReadyQueue[]队列中选择首作业添加至 DynamicQueue[]队列执行。否则先判断 BackupQueue[]队列是否为空，不为空则执行 BackupQueue[]队列中的作业，否则执行 ReadyQueue[]队列中首作业。

(4)修改 BackupQueue[]队列中的信息，此时再次判断是否有空闲 Tasktracker 节点，有则返回至(2)继续执行，否则此轮调度结束。

(5)当一轮调度结束之后，计算系统的平均负载水平，若低于系统低载，则动态地把 BackupQueue[]队列中的作业增加到 DynamicQueue[]队列中，提高系统负载水平；若高于系统高载，则减少 DynamicQueue[]队列中的执行作业数，把这些作业存入 BackupQueue[]队列中，降低系统负载水平。

(6)如果平均负载水平不属于正常范围内，那么则进入下一次的作业调度，重复(2)到(6)的步骤。

4 实验及结果分析

在实验中，选择 5 台普通 PC 机构成实验环境，分别编号为 PC1－PC5。PC 机的配置：CPU 为 Pentium(R)Dual-Core E5800，主频为 3.20GHz，内存为 2G，硬盘为 320G，操作系统为 Linux Ubuntu9.04，Java 环境为 jdk-6u24-linux-i586，Hadoop 软件版本为 Hadoop0.20.2。在实验中，我们使用 PC1 作为 Namenode 和 JobTracker 节点，剩下的 PC2－PC5 作为 Datanode 和 Tasktracker。实验的重点是对本文提出的 Three-Queue 调度算法与 Hadoop 平台上已带的 FIFO 调度算法[10]、计算能力调度算法(Capacity Schedule)[11]、公平调度算法(Fair Scheduler)[12]进行对比实验。实验中，Hadoop 中 HDFS 默认的 blocksize 大小为 64M，但由于物理设备限制，我们将其修改为 32M。

我们假设，依次执行 4 个作业任务：job1，job2，job3，job4；每个作业包含 4 个大小相同的数据块，分别是 32M、64M、128M、256M，因此每个作业执行的数据块大小总和分别为 128M、256M、512M、1024M。其他参数设置均相同，实验场景如本文第二部分假设的调度场景相同，得到作业完成时间结果如图 1 所示。

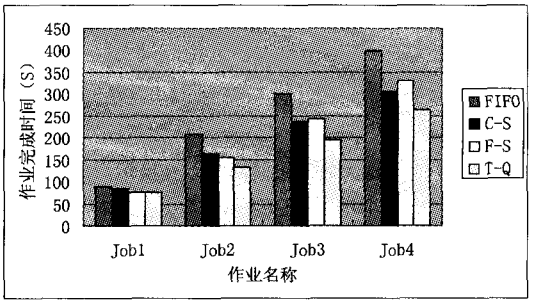

图 1 实验结果图

从图中结果比较我们可以发现，本文提出的 Three-Queue 算法相对于其它 4 个调度算法而言具有更好的执行效率。一般情况下，当数据量很小时，数据本地性问题不是非常的突出，4 个调度算法的执行效率基本相当，本文的算法效率与公平调度算法相比没有明显的优势。但随着数据量的不断增加，数据本地性问题越发突出，Three-Queue 算法的优势逐渐显现出来，执行效率明显比同类算法效果更佳。

结束语 目前 Hadoop 分布式平台受到越来越多的青睐，而其自带的调度算法已经成为其发展的瓶颈，尽管有许多其他的调度算法被提出来，但是每种调度策略都是针对特定的应用领域，有其局限性。本文提出的 Three-Queue 调度算法采用的基于优先级判定策略，考虑多方面影响因素，能够较好地解决数据本地性问题，执行效率大为提高。不足之处在于目前其主要应用于同构环境中，另外优先级计算模型也只是简单的加权求和。将其适用于异构环境，构建更精确的优先级判定计算模型将是我们主要研究方向。

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[C]// OSDI'04: Sixth Symposium on Operating System Design and Implementation. 2004: 137-150

[2] Zaharia M, Borthakur D, Sarma J S. Job scheduleing for multi-user mapreduce clusters[C]// Proceedings of the 5th European Conference IEEE. 2009: 145-161

[3] Matei Zaharia, Dhruba Borthakur and Joydeep Sen Sarma. Delay scheduling: a simple technique for achieving locality and fairness in cluster sched- uleing[C]// EuroSys '10: Proceedings of the 5th European conference on Computer systems. 2010: 265-278

[4] Polo J, de Nadal D, Carrera D. Adaptive Task Scheduling for MultiJob MapReduce Environments[C]// Proceedings of the 2010 Eighth International Conference on Grid and Cooperative

Computing IEEE. 2010:326-332
[5] Thomas Sandholm and Kevin Lai. Dynamic propor- tional share scheduling in hadoop[C]// JSSPP '10:15th Workshop on Job Scheduling Strategies for Parallel Processing. 2010:110-131
[6] Polo J, Carrera D, Becerra Y. Performance-driven task co-scheduling for mapr- educe environments[C]// Network Operations and Management Symposium(NOMS), IEEE. 2010:373-380
[7] Tian C, Zhou H, Zha L. A dynamic MapReduce scheduler for heterogeneous workloads[C]// Proceedings of the 2009 Eighth International Con ference on Grid and Cooperative Computing. IEEE Computer Society, 2009:218-224
[8] 陈全,邓倩妮. 异构环境下自适应的 MapReduce 调度[J]. 计算机工程与科学,2009:168-171
[9] 孟令芬. pc 集群作业调度算法研究[D]. 东营:中国石油大学(华东),2009
[10] Apache Hadoop[OL]. http://hadoop. apache. org/
[11] Fair Scheduler for Hadoop[EB/OL]. http://Hadoop. apache. org/common/docs/current/Fair_scheduler. html, 2010-04-15
[12] Capacity Scheduler for Hadoop[EB/OL]. http://Hadoop. apache. org/common/docs/current/Capacity _ scheduler. html, 2010-03-22

(上接第 241 页)

有的知识点关系。依据两知识点间所处位置的不同,有 3 种合并形式。

1)合并直接兄弟知识点:表示将知识点 B 的属性合并到知识点 A 上。由于多个知识点的合并可以简化为若干次两个知识点的合并过程,因而这里仅讨论两知识点合并中知识关系的变化过程。合并直接兄弟知识点要求合并两知识点的知识关系:将 B 的前导知识点集、具体知识内容添加到 A 的对应属性中,然后删除 B。

2)合并直接父子知识点:仅在父知识点 A 有唯一的子知识点 B 且子知识点 B 有唯一父知识点 A 时,方可进行合并。父子知识点合并后,A 拥有 B 所有的关联关系和父子关系,且在 A 的包含属性集中需删除与 B 有关的包含属性值,然后删除 B。

3)合并根知识点:仅 B 为根知识点时方允许合并。设知识点 A 是知识点 B 的前导知识,知识点 B 是知识点 A 的后续知识。根知识点合并后,A 拥有 B 所有的关联关系和父子关系,然后删除 B[5]。

定义 13(运算∩) 拆分知识点,用 $Split(A,B)$表示。将知识点 A 分裂为 A,B,知识点 A 的各种属性点集中删除 B 相同的属性点集,A 作为 B 的父知识点,B 作为 A 的子知识,A 和 B 中具体的知识内容由组成员协商。

定义 14(运算 §) 查看知识点,用 $Look(A,B)$ 表示。必须已查看 A 的全部前导知识并达到指定学习阈值才可以查看 A。首先,产生新知识点 $B=Create(A)$,写入达到学习阈值(比如成绩),然后 $Insert(B,A)$,形成学习的历史知识点记录。

定义 15(运算<) 确定全局序关系:用 $Global(Oa,Ob)$ 表示。确定操作 Oa 和 Ob 的全局序关系。

算法 1 sIOPT 算法

```
//HB 中有 n 个已执行知识点操作
HB:O1,O2,…,On
Ok:该站点收到的远程操作
BC:站点的当前背景知识点结构 BC=BR[On]
sIOPT(Ok){
//封装 Ok 操作的所属知识点操作,Package(Ok)中进行具体媒体文件的操作一致性处理,本文暂不讨论。
Ok1=Package(Ok)
// Ok1 是 On 的后序操作
If Ok1<On Then
// Ok1 加到 HB 尾
HB. AddTail(Ok1)
//执行 Ok1 更新背景知识点结构
BC=BC+Ok1
Return
End If
j=n
//Ok1 是的 Oj 前序操作
While(Oj<Ok1)
j--
End While
Ok1'=Ok1// 对 Ok1 所有后序操作做包含转换
For m=j+1 To n
Ok1'=IT(Ok1',Om)
End For
//将 Ok1 插入 HB 中 Oj+1的前面
HB. InsertBefore(Ok1,Oj+1)
//更新当前背景知识点结构,显示执行效果
BC=BC+Ok1'
}
```

定义 16(知识点操作转换 IT,O1,O2) 对于操作 $O1$,$O2$,如果 $O1>O2$,且 $O1 \fallingdotseq O2$,$O1$ 对 $O2$ 做包含转换 $O1'=IT(O1,O2)$,使得执行 $O2,O1'$ 和执行 $O1,O2$ 的结果一致,即:$BC[O1]+O1+O2=BC[O2]+O2+O1'$[6]。

结束语 本文解决了基于知识点树结构的协同学习中,组成员协同完成一个学习任务的过程中如何进行知识树及具体的媒体文件的一致性维护,提出了一种基于知识点操作转换的一致性算法模型。为后续的协同学习相关研究奠定了基础。

参 考 文 献

[1] 贾巍,瞿堃. 知识表示视野下网络课程知识点关系研究[J]. 继续教育研究,2008
[2] 张平安. 网络课程中的知识结构化表示方法[J]. 计算机教育,2008
[3] 朱郑州. 基于知识点本体的个性化课程组织方法[J]. 计算机科学,2009,36(12)
[4] 廖斌. 实时协同工作系统中操作转换算法综述[J]. 计算机研究与发展,2007,44(2)
[5] 肖锟. 课件知识点操作的形式化研究[J]. 计算机科学,2008,35(1)
[6] 徐向华. 可适应的实时协同编辑系统若干问题研究[D]. 2005