一种 MapReduce 实时调度算法设计及实现①

刘 吉[1], 陈香兰[1,2], 代 栋[1], 孙明明[1], 周学海[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: MapReduce 是云计算中重要的批数据处理框架, 多任务共享 MapReduce 机群并满足任务实时性要求是调度算法急需解决的问题. 提出两阶段实时调度算法, 将调度划分为任务间调度和任务内调度. 对于任务间调度, 使用抽样法和经验值法确定子任务执行时间, 利用该参数建立资源分配模型, 动态确定任务优先级进行调度; 对于子任务使用延迟调度策略进行调度, 保证计算的本地性. 实验结果显示, 两阶段实时调度算法相比公平调度算法和 FIFO 算法, 在保证吞吐量的同时能够满足任务实时性要求.

关键词: MapRuduce; 实时调度; 抽样法; 延迟调度

Design and Implementation of a Real-time Scheduling Algorithm for MapReduce

LIU Ji[1], CHEN Xiang-Lan[1,2], DAI Dong[1], SUN Ming-Ming[1], ZHOU Xue-Hai[1,2]

[1](College of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Suzhou Institute for Advanced Study, University of Scienceand Technology of China, Suzhou 215123, China)

Abstract: MapReduce is a popular batch data processing framework in cloud computing field. Sharing MapReduce cluster and meeting the deadlines of jobs is a key problem to be solved. This paper proposes a two phase real-time scheduling algorithm which separate scheduling into job scheduling and task scheduling.It uses sampling method to estimate the task excuting time so that the scheduler can make a decision on how many slots should be assigned to the job and how to calculate the job's priority. Using delay-scheduling scheme in task scheduling, the"computing locality"problem can be solved well. Experiments result shows that the scheduling algorithm implemented in this paper satisfies the job's real-time requirement as well as throughput of the cluster.

Key words: mapreduce; real-time scheduling; sampling; delay-scheduling

互联网, 尤其是移动互联网的发展加快了大数据处理存储技术的发展和变革. MapReduce[1]是由 Google 提出的一种高效分布式数据处理框架, 已成为业界广泛使用的云计算技术. 一个 MapReduce 机群性能的优劣有两个评测标准: 机群的吞吐量和任务响应时间, 而影响性能的主要因素在于调度算法的优劣. 目前常用的 MapReduce 调度算法主要有三种: 先进先出(FIFO)算法[2], 公平调度算法(FairScheduler)[2,3]和基于计算能力的调度算法(Capbility Scheduler)[2]. 其中 FIFO 算法按照任务到达顺序进行资源分配并调度, 容易导致后续任务的饥饿; 而 FairScheduler 和 Capbility Scheduler 主要针对多用户调度, 没有考虑任务的实时性问题. 本文提出一种实时调度算法, 在兼顾机群吞吐量的同时能够满足任务实时性要求. 要调度实时任务, 需要解决以下两个问题:

第一、任务执行时间预测. 一个 MapReduce 任务包含多个子任务, 子任务执行时间决定了整个任务完成时间. 当前已经提出的实时调度算法中, 有些要求用户提供该参数[4,5], 但用户通常并不知道这个时间; 文献[6]提到在任务运行前使用抽样法进行时间预测, 然而其并没有考虑 reduce 任务时间与 map 任务时间的差异, 最终导致调度精度不高. 本文使用抽样法和反

① 基金项目:江苏省产学研前瞻性联合研究项目(BY2009128)
收稿时间:2013-01-13;收到修改稿时间:2013-03-08

馈法相结合解决此问题. 实验结果显示抽样法可以得到 map 任务执行时间的准确预测值, 可直接使用; 而 reduce 预测值准确性较差, 我们仅使用该值作为初始值, 任务执行后再通过反馈方式修改.

第二、任务优先级的确定. 确定优先级是调度算法要解决的关键问题. 文献[4]和文献[6]均使用资源需求量作为优先级, 这可能导致小任务长时间得不到调度; 而文献[7]使用剩余时间作为优先级, 这会造成大任务饥饿. 本文提出一种基于持有资源比例和等待时间的动态优先级策略, 此策略既能充分体现任务的紧急程度, 又可避免饥饿问题.

基于以上两个问题的解决方案, 本文提出一种两阶段实时调度算法, 该算法包括了任务间调度和任务内调度. 在第一阶段, 我们根据任务资源占有情况计算优先级, 按优先级确定任务调度次序. 任务每次获得资源后, 其优先级降低, 防止其他任务饥饿; 在任务内调度阶段, 我们使用“延迟调度[8]”策略, 当高优先级任务无合适子任务可投入运行时, 该任务放弃此次调度, 等待合适时机. 这种方法能够在一定程度上保证计算与数据的本地化, 提高机群整体效率.

当前实时调度算法中有一类专注于解决硬实时调度[9,10]问题, 本文认为硬实时要求在 MapReduce 中并不多见, 故本文主要侧重软实时调度. 我们在实验中对两阶段实时调度策略进行评估, 结果显示在多任务环境中, 实时任务都能在规定时间内完成, 且系统吞吐量没有明显降低.

本文组织结构如下: 第一部分介绍 MapReduce 的基本知识及调度原理; 第二部分详细描述两阶段实时调度算法理论与实现过程; 第三部分介绍相关实验以及实验结果分析. 最后得出结论, 说明本文所提出的调度算法的合理性.

1 MapReduce及调度过程介绍

MapReduce 是一种分布式并行数据处理框架, 它能够隐藏底层的并行处理细节, 向用户提供简单的编程接口, 极大的简化了分布式程序的开发. MapReduce 将数据处理分成 Map 和 Reduce 两个阶段. 在 Map 阶段, 每个 Map 任务从全局文件系统或数据库中读取一个数据片段(该数据在文件系统中有多份拷贝), 使用用户自定义的 map 函数将输入数据解析成中间 key/value 对, 进行排序和分区后存储在本地; 在 Reduce 阶段, reduce 任务从每个 map 任务节点读取相应分区数据, 使用用户定义的 reduce 函数对中间 key/value 对进行处理, 将结果存储在全局文件系统或者数据库中. MapReduce 这种工作方式决定了它存在一种“本地性[8]”问题, 即 map 任务应尽量与所处理数据保持在同一节点, 这样可以减少数据拷贝; 而 reduce 任务不存在该问题. 保证本地处理能够极大提高机群效率.

在 MapReduce 中, 机群的资源以槽位(Slot)为单位进行组织, 每个槽位占用一定的内存和 CPU 资源,同时只能处理一个任务. MapReduce 是一种主-从模式框架, 主节点上运行 JobTracker, 负责监控机群, 任务调度等; 从节点上运行 TaskTracker, 负责监控任务执行, 报告进度等. TaskTracker 会定期向 JobTracker 发送心跳(heartbeat)信息, 该信息中携带本节点的资源使用情况. 主节点中的调度即发生在心跳到达时, 若 TaskTracker 报告自己有空闲资源, 则 JobTracker 使用调度算法选择一个任务发射到该节点运行. 调度的实质是槽位的分配.

Hadoop 是 MapReduce 的一个完全开源实现. 用户提交的任务称为 Job, map 和 reduce 子任务称为 Task. Hadoop 调度模块被设计为可插拔的, 为方便实验, 本文提出的调度器原型完全基于 Hadoop.

2 两阶段实时调度算法

本节介绍两阶段实时调度算法实现细节. 调度器首先获取单个任务的执行时间, 以该时间为参数进行资源分配. 更进一步, 根据作业实际获得资源情况确定优先级, 进行作业调度. 在任务调度阶段, 使用“延迟调度”策略以提升进群效率. 该算法每次获取 TaskTracker 空闲资源信息及当前处于运行状态的任务队列, 输出分配给该 TaskTracker 的任务序列.

2.1 定义

为方便后续章节算法推导, 我们首先进行如下定义:

$H=(M,R,M_n,R_n)$: 一个 Hadoop 机群, 共有 M 个 map 任务槽和 R 个 reduce 任务槽, 其中 map 槽空闲 M_n 个, reduce 槽空闲 R_n 个. $M\ge 1, R\ge 1, M_n\ge 0, R_n\ge 0$.

$Q=(A,D,m,r,m_r,m_u,r_r,r_u,m_a,m_h,r_a,r_h)$: 一个作业(Job), 提交时刻为 A, 期望执行时间为 D; 作业包含 m 个 map 任务和 r 个 reduce 任务; 正在运行的 map

任务数为 m_r，未运行的 map 任务数为 m_u；正在运行的 reduce 任务数为 r_r，未运行的 reduce 任务数为 r_u；调度器分配给该作业的 map 槽位数为 m_a，作业实际获得的 map 槽位数为 m_h；调度器分配给作业 reduce 槽位数为 r_a，作业实际获得的 reduce 槽位数为 r_h；

$m \ge 1, r \ge 0$。$m_r \ge 0, m_u \ge 0, r_r \ge 0, r_u \ge 0, m_a \ge 0, m_h \ge 0, r_a \ge 0, r_h \ge 0$.

$M = (s_m, f_m, t_{mp})$：一个 map 任务，开始时刻为 s_m，结束时刻为 f_m，已经运行的时间是 t_{mp}. 其中 $s_m < f_m$，$t_{mp} \ge 0$.

$R = (s_r, f_r, t_{rp})$：一个 reduce 任务，开始时刻是 s_r，结束时刻是 f_r. 已经运行的时间是 t_{rp}. 其中 $s_r < f_r$，$t_{rp} \ge 0$.

$\overline{\varphi}$: map 任务的平均执行时间.

$\overline{\vartheta}$: reduce 任务的平均执行时间.

2.2 任务运行时间估计

抽样法，类似于概率统计中的抽样估计. 当用户提交一个新作业(Job)后，所有任务(Task)并不是立即投入运行，而是由系统根据规则选取 u 个 map 任务和 w 个 reduce 任务作为样本提前运行. 从 JobTracker 中可以获取每个 map 和 reduce 任务的执行时间，因此我们可以获得抽样执行 map、reduce 任务的平均执行时间 $\overline{\varphi}$ 和 $\overline{\vartheta}$.

$$\overline{\varphi} = \left\lceil \frac{\sum_{i=1}^{u}(f_m^i - s_m^i)}{u} \right\rceil \tag{1}$$

$$\overline{\vartheta} = \left\lceil \frac{\sum_{i=1}^{w}(f_r^i - s_r^i)}{w} \right\rceil \tag{2}$$

由于同一作业中每个 map 任务处理的数据量基本相同(一个或几个数据块)，处理方式也完全相同(使用相同 map 函数)，故抽样运行的 map 任务与实际情况相当，其运行时间 $\overline{\varphi}$ 可直接作为真实 map 任务执行时间估计；对于 reduce 任务，多数情况下抽样运行任务数量较少，导致每个 reduce 任务所处理数据量与实际情况相差较大，若使用 $\overline{\vartheta}$ 作为估算结果会产生较大误差. 本文中，我们仅使用 $\overline{\vartheta}$ 作为一个初始参考值，在作业执行过程中，若调度器发现已有部分 reduce 任务完成，则使用该完成时间替代初始 reduce 任务预测时间，我们称此种方法为反馈法. 反馈法有助于调度器掌握精确的任务预测时间，为资源分配的准确性提供保证.

2.3 动态资源分配

为了保证作业在期望时间 D 内完成，系统需要监测作业进度并通过资源重分配控制作业的推进速度. 设作业 map 阶段执行时间为 t_m，reduce 阶段执行时间为 t_r，则:

$$t_m + t_r \le D(D - r.\overline{\vartheta} \le t_m \le D - \overline{\vartheta}, D - m.\overline{\varphi} \le t_r \le D - \overline{\varphi}) \tag{3}$$

若作业获取到 r 个 reduce 任务槽，则其 reduce 任务可并行完成，需要时间 $\overline{\vartheta}$，故 t_m 最大值为 $D - \overline{\vartheta}$；相反，若作业只能得到一个 reduce 任务槽，则其 r 个 reduce 任务只能串行执行，需要时间为 $r.\overline{\vartheta}$，故 t_m 的最小值为 $D - r.\overline{\vartheta}$. 同理可确定范围 t_r. 在满足约束条件下，本文参考机群当前状态确定 t_m、t_r 的分配. 若当前空闲 map 槽位较多，则分配较小 t_m (因为可以分配较多的 map 任务槽以提高 map 任务并行度，缩短其执行时间)；反之分配较大 t_m. 作业提交后，调度器为其分配初始资源. 由公式(1)(2)(3)及作业 Q 定义，可推导出为 map 任务分配的初始资源数 N_m 和为 reduce 任务分配的初始资源数 N_r 如下:

$$N_m = \left\lceil \frac{m.\overline{\varphi}}{t_m} \right\rceil \tag{4}$$

$$N_r = \left\lceil \frac{r.\overline{\vartheta}}{t_r} \right\rceil \tag{5}$$

考虑到 reduce 任务的估算时间 $\overline{\vartheta}$ 比实际值小(样本 reduce 任务处理数据量较小)，故在分配 t_m 与 t_r 时，可适当减小 t_r，以平衡 N_r 值.

作业运行过程中，可能发生节点失败，网络延迟等故障导致作业推进速度低于预期，因此需要对作业资源进行动态分配(一般是增加)，以保证时间要求. 由于 MapReduce 框架规定所有 map 任务结束后才能开始 reduce 任务，我们将作业划分为两个阶段: map 阶段和 reduce 阶段. 我们的目标是: map 阶段时间不超过 t_m，如果无法保证，则 map 阶段要占用 reduce 阶段时间，系统需要增加 reduce 任务资源；reduce 阶段时间不超过 $A + D$，若超过，则作业将不能按时完成，调度器需分配作业全部所需资源以保证作业尽快完成. 若作业处于 map 阶段，重新分配的资源数(map 槽位) N_{mr} 为:

$$N_{mr} = \begin{cases} \left\lceil \dfrac{m_u.\overline{\varphi}}{t_m - (curr\_time - A)} \right\rceil & (map\text{阶段未超时}) \\ m_u & (map\text{阶段超时}) \end{cases} \tag{6}$$

如果 map 阶段时间未超过 t_m，使用公式(6)中的第一个公式(公式(4)的扩展形式)计算 N_{mr}；反之，则分配最大数目的任务槽 m_u，以尽快完成此阶段. 若作业处于 reduce 阶段，我们使用反馈方法重新设置 reduce 任务估算时间，利用新值进行资源分配. 若部分 reduce 任务已完成，设其完成时间为 ϑ，则我们使用 ϑ 作为新的 reduce 时间估计值. 重新分配的资源数(reduce 槽位) N_{rr} 为:

$$N_{rr}=\begin{cases}\left\lceil \dfrac{r_u.\vartheta}{A+D-curr\_time}\right\rceil & (\text{部分reduce任务完成})\\ \left\lceil \dfrac{r_u.\overline{\vartheta}}{A+D-curr\_time}\right\rceil & (reduce\text{任务均未完成})\end{cases} \tag{7}$$

通过以上资源分配方法，如果作业均已获得足够任务槽，而系统资源仍有较多剩余，我们设定一个保留值，将超过此值的资源按作业所需资源比例进行分配. 这种空闲资源利用策略既可保证已提交作业能快速完成(已提交作业得到了更多的资源)，又能确保新作业可及时响应(系统有部分保留资源).

2.4 调度策略

本文提出的实时调度策略从两个层次实现：作业(job)调度和任务(task)调度. 作业调度目标是高优先级作业优先获得资源(槽位)；任务调度目标是从作业中选出一个或几个最合适的任务，发射到任务槽中执行. 这里的“最合适”指可以尽可能的选择具有“本地性”的任务.

2.4.1 作业调度

本文使用“缺额程度”表示作业优先级. “缺额程度”指的是作业尚需的资源数与应获得的资源数的比值. 对于处在 map 阶段作业，按如下方式计算优先级:

$$PRI_m=\frac{m_a-m_h}{m_a} \tag{8}$$

其中 $m_a=N_{mr}$，即调度器分配给作业的任务槽数；m_h 表示作业实际已经获取的 map 任务槽数. 同理，处于 reduce 阶段的作业的优先级为:

$$PRI_r=\frac{r_a-r_h}{r_a} \tag{9}$$

其中 $r_a=N_{rr}$，即作业应获得的任务槽数；r_h 表示已获得的 reduce 任务槽数.

作业根据所处阶段不同按照上述计算方式确定优先级进行调度. 可以预测，当作业一个阶段(map 阶段或 reduce 阶段)即将执行完毕时，其优先级会越来越小(因为“缺额”已经很小). 若新作业不断提交，将会使本作业长时间无法获得资源，导致饥饿或饿死. 为防止此种现象发生，我们采用一种辅助优先级调节方法：调度器监测作业当前阶段的执行时间，若发现当前阶段即将超时或已超时，则每次心跳到来进行调度时，调度器都增加其优先级. 使用这种方法可保证即将完成的任务即使缺额程度很小，也能获得资源继续执行.

调度器每次按照优先级为作业分配资源，若出现相同优先级情况，可随机选取其中一个作业. 随机选择是合理的，因为调度频率很高，每次调度后会重新计算本作业优先级(降低)，下次心跳到达进行调度时，之前与之具有相同优先级的作业将会得到资源.

2.4.2 任务调度

由于只有 map 任务存在“本地性”特征，所以任务调度主要针对 map 任务调度. 分配资源时，调度器应优先考虑具有“本地性”和“部分本地性”特征的 map 任务. 若不存在这样的任务，我们采用一种“延迟调度”的策略，即暂时将资源分配给其他合适任务，待满足本地性时再调度本任务. 为实现延迟调度，我们为每个作业设定一个 skip 属性，当某次该作业优先级最高，但不具备“本地性”特征时，作业让出本次资源，skip 自动增长. 当 skip 值增长到预先设定的阈值时，该作业放弃“本地性”要求，随机选取一个任务投入运行. “延迟调度”机制能够提高数据本地处理的比例，提高机群效率.

2.5 调度算法描述

根据以上推导及调度器工作原理，本文基于 Hadoop 实现了两阶段实时调度算法，算法描述如下:

Algorithm TwoPhaseRTScheduler

Input:A job queue(feed by the hadoop framwork) and a TaskTracker status which indicates the free resources of the node.

Output: The list of tasks which should be assigned to the given TaskTracker.

Begin

1. Separate the job queue into map-phase queue and reduce-phase queue.
2. If some jobs are new submitted,caculate the priority. /*新提交的任务需要计算初始任务优先级*/
3. If the TaskTracker has free map slots then

/*某工作节点有空余 map 任务槽，按以下策略分配*/

3.1 Sort the map-phase queue according the priority.

3.2 Traverse the map-phase queue to find a suitable task for each slot.

/*遍历该优先队列，按以下策略决定获取资源的任务*/

3.2.1 If the job's status is SAMPLING and need more samples,select tasks from this job and add it to task list,recalculate job's priority and go to step 3.2.

/*新提交任务优先获得资源进行抽样执行*/

3.2.2 If the job has locality tasks,add the tasks to the task list, ,recalculate job's priority and go to step 3.2.

/*若任务具有本地性，则该任务获得本次资源*/

3.2.3 If the job has no locality tasks,delay scheduling this job,go to step 3.2.

/*不具有本地性的任务执行延迟调度策略*/

4. If the TaskTracker has free reduce slots then

/*reduce 调度每次直接从优先队列头部取得一个任务获得该任务槽*/

4.1 Sort the reduce-phase queue according the priority.

4.2 Select a task from the first job in queue and add it to task list,recalculate the job's priority.

5. Return the task list.

/*返回该工作节点获得的任务列表*/

End

本算法输入是一个任务队列和表示每个工作节点状态的 TaskTracker Status，这两个参数均由 Hadoop 框架提供. 算法输出是分配给每个任务节点的任务列表. 在 JobTracker 中，每当心跳到达时，框架触发上述任务调度算法进行资源分配. 值得说明的是：在 4.2 节我们每次只分配一个 reduce 任务给该 tasktracker，这样有利于保证机群的负载均衡.

3 实验与结果分析

本节介绍两阶段实时调度算法相关实验. 实验一证明使用抽样方法估算 map 任务执行时间可行，而 reduce 任务执行时间则有较大误差；实验二实现并测试了本文提出的“两阶段实时调度(TwoPhaseRTS-cheduler)算法”，通过与公平调度算法、先进先出调度算法对比，证明该算法在相同情况下既能满足任务实时性，又能保证系统吞吐量.

本文所有实验均在 Hadoop 平台上完成. 我们使用 Hadoop 发行包自带的 wordcount 和 terasort 作为测试程序. wordcount 是一个单词计数程序，能够代表一类典型的 mapreduce 应用，如搜索引擎中常用的倒排索引建立. terasort 是 TB 级数据排序程序，常用来测试机群性能，在分布式数据处理中应用也较为广泛. 为保证实验结果正确性，我们使用的每个数据均是程序四次运行的平均数值.

本文实验使用具有 6 个节点的 Hadoop 机群，其中一个节点作为 master，其余五个作为 slave. 机群是同构的，每个节点均采用 Intel(R) Xeon(R) CPU，双核，6GB 内存，1TB 硬盘，运行 64 位 Fedora17 系统，JRE 版本为 1.7.0_05，每个节点都部署 hadoop1.0.3. 根据实际情况，我们在每个 slave 节点上配置 2 个 map 槽位，2 个 reduce 槽位，并设定 hdfs 数据块大小为 64MB. 所有节点通过 1GB 以太网交换机相连.

3.1 任务运行时间估计

我们单独运行 wordcount 和 terasort 程序，选择不同数目 map 任务，而 reduce 任务设定为一. 由图 1、图 2 可见：随着 map 任务数量增加，map 任务平均处理时间并没有较大的变化.

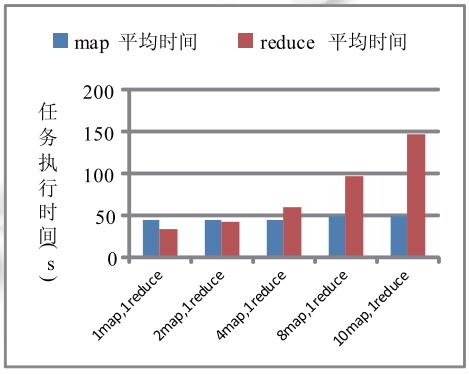

图 1 wordcount 独立执行时间

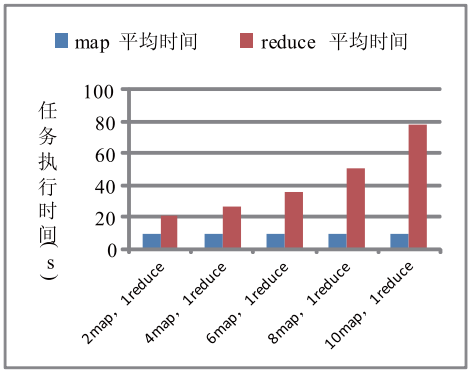

图 2 terasort 独立执行时间

这说明在一个作业中, 使用少量 map 任务执行时间作为 map 任务执行时间预测是合理的; 而对 reduce 任务, 情况则不然. 随着 map 数量增加, reduce 任务处理时间也随之增加, 故不能简单的使用少量 reduce 任务执行时间作为最后的预测值. 另一方面, 由图 3、图 4 可见, 若可保证 map、reduce 任务数量确定比例, 那么 reduce 任务执行时间变化不大, 可直接作为预测值. 如图 3 中 reduce 执行时间在 42s～58s 间. 通过以上实验我们可以得出以下结论: ①使用抽样法能够准确预测 map 任务执行时间; ②若可按原任务数量比例进行抽样, 则 reduce 任务执行时间预测精度较高. 然而实际情况下, map 任务数量通常远远大于 reduce 任务数量, 如 google 常用 40: 1 比例, 为降低抽样执行时间和资源消耗, 应尽量减小抽样样本. 所以实际操作中我们可根据情况选择近似比值(如 10: 1)进行抽样, 再使用 2.3 节中反馈方法修改 reduce 任务时间估计值, 重新计算作业资源.

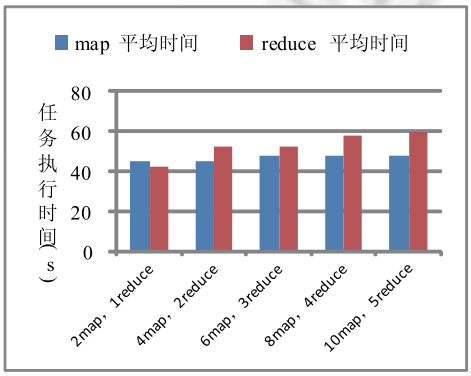

图 3　wordcount 独立执行时间(多 reduce)

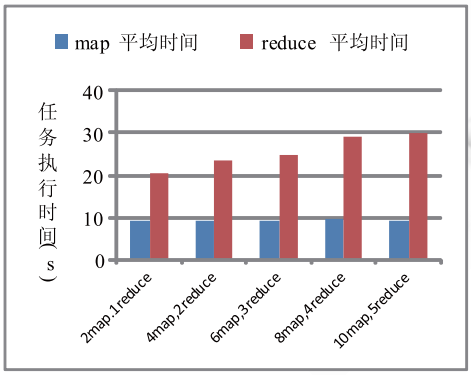

图 4　terasort 独立运行时间(多 reduce)

3.2 调度算法验证

在此实验中, 我们分别使用本文提出的两阶段实时调度算法(TwoPhaseRTScheduler), 先进先出调度算法(FifoScheduler), 公平调度算法(FariScheduler)调度两个 wordcount 程序(10map, 5reduce)和两个 terasort 程序(10map, 5reduce), 作业的提交顺序为 wordcount1, terasort1, wordcount2, terasort2. 我们设定 wordcount 程序的 deadline 为 240s(单独运行时间为 120s), terasort 程序的 deadline 为 120s(单独运行时间为 60s). 由于本实验测试程序任务数量较少, 四个测试程序中作为样本提前运行的 map 和 reduce 任务数分别为 1.

由图 5 中三种调度策略对比可见, Fifo 调度策略只能保证 wordcount1 程序在规定时间内完成, 而其余三个均超过 deadline. 这是由于 Fifo 调度策略按提交顺序优先满足先提交作业的资源需求. 本例中, wordcount1 首先提交, 它占用了系统中所有的 map 任务槽, 导致其他程序被阻塞而延迟; 作为对比, 使用公平调度策略, 两个 terasort 程序能够很快完成, 而两个 wordcount 程序略微超过了 deadline. 整体上实时性较 Fifo 强. 公平调度策略使用“资源池”保证了每个作业都会获得资源, 避免了单个任务过度使用资源导致其他任务阻塞; 而使用两阶段实时调度策略, wordcount 执行时间较 FairScheduler 短, terasort 执行时间较 FairScheduler 长, 但均在其 deadline 之前. 这是由于该策略使用任务截止时间计算优先级, 并在任务执行过程中动态修改该优先级, 尽量保证任务能在规定时间内完成, 任务的紧急程度此消彼长, 最终所有任务都获得了合理的资源. 从吞吐量角度看, TwoPhase RTScheduler 在 240s 内完成所有作业, 在 120s 内完成两个 terasort 程序. 相同时间内 FifoScheduler 完成的作业数为 1, 0; FairScheduler 完成的作业数为 3, 2. 由此可见, 通过实时性改造, 本文提出的算法在以作业 deadline 为时间点的吞吐量方面表现较好, 这也具有必然性, 因为该调度算法相比 Fifo 算法只是增加了计

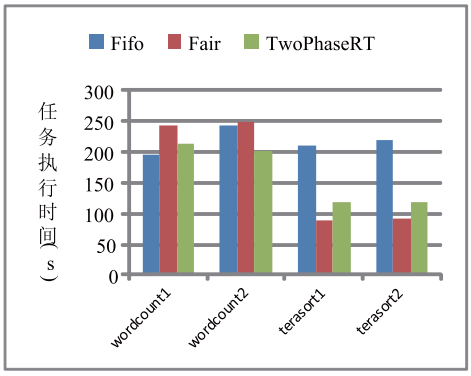

图 5　三种调度策略对比

算优先级环节, 以此来决定获得资源的任务, 而该过程的时间消耗极少. 实验中我们还发现, FifoScheduler 在同一时刻只调度一个作业, 而 FariScheduler 和 TwoPhaseRTScheduler 则能保证四个作业同时推进. 这是因为本文算法通过优先级改变保证任务交替获取资源, 宏观上表现为多任务同时执行. 这也说明两阶段实时调度策略可适用于多任务环境.

4 结语

为了解决 MapReduce 实时调度问题, 本文提出了一种软实时调度算法, 通过任务时间估计和优先级动态修定, 并结合延迟调度策略, 在保证机群效率的同时实现实时调度. 实验结果表明, 本文提出的实时调度策略在实时性方面优于现有调度算法, 同时能保证系统吞吐量不会降低.

参考文献

1 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. ACM,New York,NY,USA.2008: 107–113.

2 White T,周敏奇,王晓玲,金澈清,钱卫宁.Hadoop 权威指南.第 2 版.175–177.

3 Zaharia M. Hadoop Fair Scheduler Design Document.2009.

4 Polo J, Carrera D, Becerra Y, Torres J, yguadé E, Steinder M, Whalley I. Performance-driven task co-scheduling for Map Reduce environments. Network Operations and Management Symposium (NOMS),2010 IEEE.373–380.

5 Kc K, Anyanwu K. Scheduling Hadoop Jobs to Meet Deadlines. Cloud Computing Technology and Science (CloudCom), 2010 IEEE 2nd International Conference. 388–392.

6 Dong XC, Wang Y, Liao HM. Scheduling Mixed Real-Time and Non-real-Time Applications in MapReduce Environment. 2011 IEEE 17th International Conference on Parallel and Distributed Systems.

7 Dou AJ, Kalogeraki V, Gunopulos D, Mielikainen T, Tuulos V. Scheduling for Real-Time Mobile MapReduce Systems. Proc. of the 5th ACM international conference on Distributed event-based system, 2011.347–358.

8 Zaharia M, Borthakur D, SenSarma J, Elmeleegy K, Shenker S, Stoica I. Job scheduling for multi-user Map Reduce clusters. EECS Department, University of California, Berkeley, Tech. Rep.UCB/EECS-2009-55, Apr2009.

9 Phan LTX, Zhang ZY, Loo BT, Lee I. Real-time Map Reduce Scheduling. Technical Report No. MS-CIS-10-32, University of Pennsylvania, 2010.

10 Davis RI, Burns A. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multi-processor systems. Technicalreport, Dept. of Computer Science, University of York, 2009.

(上接第 90 页)

参考文献

1 王志文,郭戈.移动机器人导航技术现状与展望.机器人,2003,25(5):193–197.

2 徐德,邹伟.室内移动服务机器人的感知、定位与控制.北京:科学出版社,2008.

3 张毅,罗元,郑太雄.移动机器人技术及其应用.北京:电子工业出版社,2007.

4 吴刚,唐振民.单目式自主机器人视觉导航中的测距研究.机器人,2010,32(6):828–832.

5 张广军.机器视觉.北京:科学出版社,2005.

6 张法全,路立平,沈满德,陈良益,崔光照.单目视觉目标距离测量方法研究.光子学报,2009,38(2):453–456.

7 Lee WL, Hsieh KS. A robust algorithm for the fractal dimension of images and its applications to the classification of natural images and ultrasonic liver images. Signal Processing, 2010,90(6):1894–1904.

8 Liu YJ, Luo X, Xuan YM, Chen WF, Fu XL. Image Retargeting Quality Assessment Computer Graphics Forum, 2011,30(2):583–592.

9 郭磊,徐友春,李克强,连小珉.基于单目视觉的实时测距方法研究.中国图象图形学报,2006,11(1):74–81.

10 王荣本,李斌,储江伟,纪寿文.公路上基于车载单目机器视觉的前方车距测量方法的研究.公路交通科技,2001,18(6):94–98.