基于任务分类的延迟调度算法①

高正九, 郑 烃, 辛 波, 王 嵩

(中国科学技术大学 信息科学技术学院, 合肥 230027)

摘 要: MapReduce 已经成为主流的海量数据处理模式, 任务调度作为其关键环节已受到业界广泛关注. 针对已有的延迟调度算法存在的问题, 即建立在任务都是短任务的理论假设有一定限制, 当节点处理不同长度的任务时算法性能严重下降和基于静态的等待时间阈值不能适应不同用户的作业需求, 提出了一种基于任务分类的延迟调度算法. 该算法通过给不同长度的任务设置不同的等待时间阈值, 以适应不同作业的响应需求. 通过分析各动态参数, 根据所建任务模型调整任务的等待时间阈值. 仿真验证该算法在响应时间及负载均衡性方面优于已有的延迟调度算法.

关键词: 云计算; 延迟调度; 数据本地性; 任务分类; MapReduce

Delay Scheduling Algorithm Based on Task Classification

GAO Zheng-Jiu, ZHENG Quan, XIN Bo, WANG Song

(School of Information and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: MapReduce has become a mainstream mass data processing mode, as its crucial part, the scheduler has received extensive concerns of the industry. There are two deficiencies in the current delay scheduling algorithms. Firstly, a limitation of these policies is that all the tasks to be processed should be short as assumed, the performance of the algorithms declined serious when servers handle the tasks of different lengths. Secondly, delay scheduling algorithms based on static waiting time threshold, cannot adapt to the different user needs. To address this issue, this paper proposed a delay scheduling algorithm based on task classification. It adjusted tasks waiting time threshold dynamically according to the information of the different lengths. It shows that this algorithm outperforms previous delay scheduling algorithms in term of the job response time and load balance of the node.

Key words: cloud computing; delay scheduling; task classification; data-locality; MapReduce

近年来, 互联网应用平台的数据量日趋庞大, 为了解决对海量数据的存储与处理, 并利用有效的资源管理与调度策略提高处理效率, 云计算海量数据处理平台应运而生, 如 Dryad[1]、Hadoop[2]等. 在这些平台上, 需要并发执行多个作业. MapReduce 正是在这一背景下产生的, 由 Google 工程师发起设计并实现用来处理大规模数据的分布式编程框架[3-5], 其后开源项目 Hadoop 实现了这一框架. 基于 MapReduce 的服务正变得越来越普及[6].

MapReduce 中的任务调度功能就是将用户提交的作业划分为多个任务, 由作业服务器分配到空闲的任务服务器上执行. 但是在 Hadoop 框架中, 却存在着任务和所需要的数据资源可能位于不同的物理位置[5,7], 当任务所需要的数据处于不同的位置时, 需要迁移数据, 此即所谓的数据非本地化问题[8-14]. 由于当数据和计算任务处于同一节点时, 计算效率才能得到保证, 因此, 数据的本地化程度是决定Hadoop框架下云计算效率的重要因素. 然而在多用户共享集群的环境下, 资源的共享使得作业之间存在竞争性, 导致任务能够运行在包含其所需数据的节点上的可能性降低, 引发

① 基金项目:国家发改委 CNGI 课题(CNGI-09-03-14)
收稿时间:2014-01-21;收到修改稿时间:2014-03-03

了数据非本地化的问题, 导致常常需要从远处节点拷贝数据, 由此带来的网络带宽浪费、计算效率低下等问题非常突出. 如何提高数据本地化程度, 节省网络带宽, 提高 MapReduce 任务的执行效率, 对开源平台 Hadoop 至关重要.

为了同时保证数据的本地性与公平性, 文献[9]提出了云计算平台上的延迟调度(delay scheduling)算法: 当一个作业需要被调度时, 如果不能保证数据本地性, 则不调度该作业, 而使其等待一段时间. 由于采用上述的等待策略, 延迟调度算法在基本保证作业公平分享资源的同时, 大幅提高数据本地性. 但是数据本地性并不是越高越好, 一方面, 数据本地性的提高会带来等待开销和公平性的牺牲; 另一方面, 在任务种类多样的应用场景下, 系统对数据本地性的需求变得不同. 因此确定合理的等待时间阈值成为影响算法性能的关键. 针对以上两方面, 本文提出了根据任务种类设定等待时间阈值的算法.

1　研究背景

1.1　云计算处理平台

在Hadoop集群搭建的云计算处理平台中, 数据文件被存储在 HDFS[15]中, 每个文件被划分成若干个大小相等的文件块, 默认大小为 64M. 为了保证文件的可靠性, 每一个文件块都存在多个文件块副本, 默认副本数为 3 份. 当处理一个文件时, 管理节点将数据处理作业划分成若干个相互独立的任务, 每一个任务处理一个数据块, 当所有的任务都完成时, 该作业才能完成. 如图 1 所示, 作业 1 分为 5 个任务, 只有当 5 个任务都完成之后, 作业 1 才算完成.

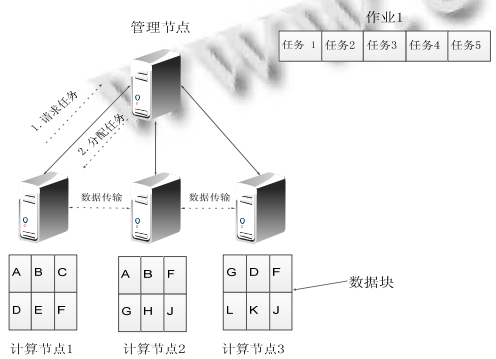

图 1　Hadoop 集群体系架构

同样数据本地化也如图 1 所示, 如果任务 2 需要数据块 G, 但是却运行在计算节点 1 上, 这时候就会导致数据非本地化的问题, 此时任务 2 被称为非本地任务. 如果任务 2 恰好运行在计算节点 2 上, 那么称任务 2 为本地任务.

总结来讲, 任务从计算节点的本地磁盘或者通过网络读取输入数据. 其中从本地磁盘读取输入数据的任务称为数据本地任务, 通过网络读取输入数据的任务称为数据远程任务. 通常而言, 数据本地包括节点本地和机架本地, 节点本地指运行任务的节点与包含任务所需数据的节点是同一个节点, 机架本地指运行任务的节点与包含任务所需数据的节点在同一个机架内.

1.2　延迟调度算法

延迟调度算法的基本思想是: 公平性方面, 利用 Max-Min 公平调度算法[16]达到统计复用. 数据本地性方面, 若当前空闲的计算节点的本地磁盘中没有队首作业所需的数据, 则先调度其他作业而让队首作业等待, 若队首作业的等待的时间超过阈值, 则立刻调度队首作业而不再等待.

延迟调度算法已经在Hadoop中实现. 它通过临时牺牲一定的公平性来提高数据本地性. 针对不同级别的数据本地性, 存在不同的时间等待阈值. 在最新的 Hadoop 版本中, 针对节点本地性和机架本地性的等待阈值均为 5 秒. 延迟调度算法的执行过程如算法 1 所示.

算法 1　延迟调度算法(DS)

```
输入: 当前到达的空闲计算节点 n
将作业按照运行任务数升序排列并保存
for j in jobs do
  //查找作业 j, 找到节点 n 的本地任务执行
  if j has unlaunched task t with data on n then
    launch t on n
    set j.wait = 0
  //如果找到作业 j 拥有未执行的非本地任务
  else if j has unlaunched task t then
    //如果等待时间超过阈值也立即执行
    if j.wait ≥ T_nodeWait then
        launch t on n
    else
        set j.wait = j.wait+1
    end if
  end if
end for
```

1.3 问题的提出

在上述算法 1 中提出的延迟调度算法是基于任务都是短任务的假设, 同时等待时间是静态的. 在 Hadoop 的最新版本中为了保证节点本地和机架本地的等待时间阈值都是 5 秒($T_{nodeWait} = T_{rackWait} = 5s$). 等待时间阈值是影响延迟调度算法性能的关键, 过长的等待时间将造成等待开销, 而等待时间不足将产生网络传输开销. 针对极短任务的时候, 假设任务运行时间小于 5 秒, 片面的追求数据本地化显得不合时宜, 而针对长任务时, 本身任务执行时间就比较长, 再让其较长时间的等待, 会让响应时间显得更加的漫长. 当 MapReduce 同时服务于极短任务、短任务、长任务的场景时, 基于静态等待阈值的延迟调度算法无法很好的发挥集群最大性能, 导致作业响应时间过长, 用户体验变得比较差.

针对以上问题, 有必要对延迟算法进行一些研究改进.

2 改进的延迟调度算法

2.1 任务的定义

定义 1. 任务的长度. 一般任务的长度定义为任务执行的时间, 但是一般Hadoop中的任务无法提前预估其执行时间, 这里定义任务的长度为任务本身的文件大小[17]. 任务 i 的长度记做 F_i .

定义 2. 任务的长度系数. 对于每一个 Map 的任务, HDFS 中文件块大小是最大的输入单元. 我们不妨记 HDFS 中文件块大小为 F_b , 任务的长度系数为任务的长度与文件块大小的比值. 那么任务 i 的长度系数表示为: $\beta_i = F_i / F_b$, 这里 $0 < \beta_i < 1$.

定义 3. 任务的分类. 根据任务的长度系数, 我们将任务分为三类: 极短任务, 短任务, 长任务. 分类的方法如下:

$$\text{任务种类} = \begin{cases} \text{极短任务}, & 0 < \beta_i < k_1 \\ \text{短任务}, & k_1 \le \beta_i < k_2 \\ \text{长任务}, & k_2 \le \beta_i < 1 \end{cases} \tag{1}$$

其中 k_1 和 k_2 表示判别任务长短的阈值, 其中一般 $k_1 \le 0.01$, k_2 需要根据实际情况来赋值.

定义 4. 任务的等待时间. 这里任务的等待时间分为节点本地化等待时间和机架本地化等待时间, 任务 i 的等待时间记做 T_{ij} , 其中 $j = 1$ 表示节点本地化等待时间, $j = 2$ 表示机架本地化等待时间. 我们根据定义 3 的分类来定义各类任务的等待时间, 针对极短任务而言, 定义如下:

$$T_{ij} = \begin{cases} 0, & j = 1 \\ TW_j * \dfrac{(1-\beta_i)}{l}, & j = 2 \end{cases} \tag{2}$$

其中 l 是正整数, 防止极短的任务在机架本地化过程中等待时间过长.

针对短任务和长任务而言, 定义如下:

$$T_{ij} = TW_j * (1-\beta_i) \tag{3}$$

其中 TW_j 表示等待时间上限值. Hadoop 中现在静态等待时间均为 5 秒, 这里我们可以取 $TW_1 = TW_2 = 5s$.

现举例说明, 如果系统中存在极短任务, 运行时间为 2 秒. 如果当前不存在节点本地化的计算节点给极短任务服务, 这时候极短任务需要等待 5 秒之后才能继续有机会执行, 花费的时间至少 7 秒, 而这个时候如果针对短任务能够机架本地化优先的话, 这时候能够立即执行短任务, 只需要花费 2 秒. 假设上述(2)式中, $l = 5, \beta_i = 0.01$, 此时等待时间为 0.99 秒, 短任务至多需要 2.99 秒即可获得返回. 如果系统中存在短任务和长任务需要执行还是根据之前的算法来确定延迟调度策略, 唯一的不同就是根据任务的长短来确定等待时间阈值. 假设长任务长度系数为 0.8, 那么这时候长任务的等待时间为: 5 秒*(1-0.8) = 1 秒. 这个时候长任务只需要等待 1 秒, 总共最多需要等待 2 秒即可执行, 也就是说至多 22 秒就可以返回结果, 而不会让长任务过长时间的等待.

2.2 基于任务分类的延迟调度算法的描述

基于任务分类的延迟调度算法(TCDS)由确定任务长短、分析任务所属类别、调整任务等待时间阈值三个阶段组成: 调用 Max-Min 公平调度算法确定作业优先级; 通过任务类别求解各任务等待时间阈值. 基本步骤如下:

a) 调用 Max-Min 公平调度算法确定作业优先级.

b) 通过任务的长度将任务分类.

c) 根据任务的长度来计算各任务的等待阈值.

d) 根据任务类别和任务等待阈值来调度任务.

基于任务分类的延迟调度算法的伪代码描述如下:

算法 2　基于任务分类的延迟调度算法(TCDS)

输入: 当前到达的空闲计算节点 *n*
将作业按照运行任务数升序排列并保存
for *j* in *jobs* do
  //查找作业 *j*, 找到节点 *n* 的本地任务执行
  if *j* has unlaunched task *t* with data on n then
    launch *t* on *n*
    set *j.wait = 0*
  //如果找到作业 *j* 拥有未执行的非本地任务
  else if *j* has unlaunched task *t* then
    //如果是极短的任务, 立即调度
    if l_t= 1
      launch *t* on *n*
    //如果等待时间超过阈值也立即执行
    else if $j.wait \geq T_t$ then
      launch *t* on *n*
    else
      set *j.wait* = *j.wait*+1
    end if
  end if
end for

3　仿真实验及分析

为了分析本文中所提出的 TCDS 算法的性能, 下面将改进的延迟调度(TCDS)算法与原始的延迟调度(DS)算法、无延迟的 FIFO 算法进行比较. 通过 MATLAB 仿真实验, 分析了算法的性能.

3.1 实验设置

采用 MATLAB 仿真 Hadoop 环境, 设置 50 个计算节点, 每个数据块大小为 64 M, 采用 Hadoop 默认的三个副本策略, 副本分配矩阵随机给出. 这里设置任务种类为: 极短任务、短任务和长任务. 这里设置初始等待时间阈值为: . 其中对于极短任务根据公式(2)设置如下参数:

其中单个极短任务的完成时间大概都在 2 秒至 5 秒, 短任务完成时间大致都在 20 秒, 长任务完成时间大致都在 60 秒左右. 根据任务所占比例分别设置五组实验. 具体设置如表 1 所示:

表 1　任务分配数目表

组号	极短任务数	短任务数	长任务数
1	50	50	10
2	10	50	50
3	30	30	30
4	50	10	10
5	10	10	50

3.2 实验结果

实验组 1: 本组实验的目的是验证 TCDS 算法是否能够获得很好的任务响应时间. 实验中根据上述表 3, 分别运行五组实验, 但是每次每种任务所占的比例不同. 这里将实验结果与原始的 DS 算法和 FIFO 算法进行比较, 如图 2 所示.

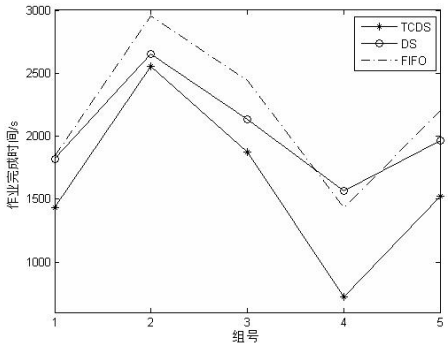

图 2　作业完成时间比较

从图 2 可以看到, TCDS 算法相比原始的 DS 算法和 FIFO 算法, 当整个集群中调度的极短任务数越多的时候, 获得的任务响应时间越好, 当整个集群中调度的长任务数越多的时候, 任务响应时间获得一定程度的提升, 当整个集群中调度的短任务数居多的时候, TCDS 算法和 DS 算法相当, 均优于 FIFO 算法. 这种现象的原因在于, TCDS 算法在处理极短任务居多的时候, 首先会本地机架优先, 这样节省了很多的等待时间, 因为极短任务本身运行时间就很短, 如果等待时间很长反而超过了极短任务本身运行的时间就造成了任务响应时间的瓶颈. 在 TCDS 算法中解决了这个瓶颈. 对于长任务居多的时候而言, 如果任务越长, 在 TCDS 算法中等待的时间反而越短, 会尽快优先让长任务执行, 这时候对于整体任务响应时间会获得一定程度上的提升. 对于短任务居多的情况, 保持和 DS 算法相当的任务响应时间.

实验组 2: 本组实验的目的是验证 TCDS 算法是否能够获得比较好的数据本地性. 实验根据上述表 3, 分别运行五组实验. 最后将实验结果与原始的 DS 算

法和 FIFO 算法进行比较, 如图 3 所示.

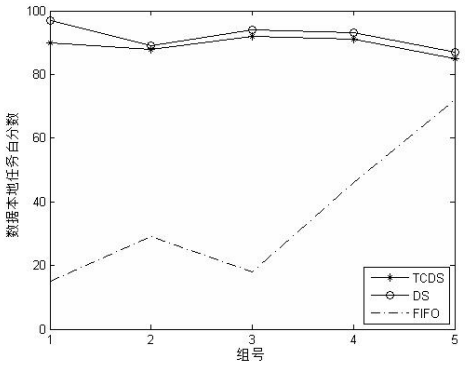

图 3 数据本地性比较

从图 3 中可以看到, TCDS 算法在极短任务较多的时候数据本地性会比 DS 算法差一些, 和 DS 算法一样都远远好于 FIFO 算法的数据本地性. 当短任务比较多的时候, TCDS 算法和 DS 算法本地性基本相当. 当长任务比较多的时候, TCDS 算法和 DS 算法数据本地性都下降了. 这是由于当极短任务占多数的时候, 牺牲掉一定的数据本地性来保证极短任务能够无需等待很长时间即可执行; 当短任务占大多数的时候, TCDS 算法会退化为 DS 算法, 所以此时它们的数据本地性基本相当; 当长任务比较多的时候, 数据节点迟迟无法空闲出来, 所以 TCDS 算法和 DS 算法均存在数据本地性下降的情况.

综上所述, TCDS 算法能够在牺牲极少的数据本地性上获得更好的任务响应时间, 从而可以缩短用户作业的响应时间, 大大提高了整个集群的利用率和服务质量.

4 结语

本文提出了基于任务分类的延迟调度算法, 该算法由确定任务长短、分析任务所属类别和调整任务等待时间阈值三个阶段组成. 该算法通过调用 Max-Min 公平调度算法确定作业优先级并通过任务类别设定各任务等待时间阈值. 为了对算法性能进行测试搭建了实验平台并进行了仿真实验. 实验结果表明, 该算法能够在牺牲极少的数据本地性上获得更好的任务响应时间, 从而可以缩短用户作业的响应时间, 大大提高了整个集群的利用率和服务质量. 下一步的工作是基于本文中的算法, 考虑网络带宽等因素, 进一步对算法进行优化.

参考文献

1 Isard M, Budiu M, Yu Yuan, et al. Dryad: distributed data-parallel programs from sequential building blocks. Proc. of ACM SIGOPS/EuroSys European Conference on Computer Systems. New York. ACM Press. 2007. 59–72.

2 Hadoop. http: //hadoop.Apache.org [2012-03-12].

3 Dean J, Ghemawat S. MapReduce: simplified data processing on large cluster. Communications of the ACM, 2008, 51(1): 107–113.

4 Ralf L. Google's MapReduce programming model-revisited. Science of Computer Programming, 2008, 70(1): 1–30.

5 Dean J, Ghemawat S. MapReduce: a flexible data processing Tool. Communications of the ACM, 2010, 53(1): 72–77.

6 Amazon mapreduce. http://aws.amazon.com/

7 Shvachko K, Kuang HR, Radia S, et al. The Hadoop distributed file system. Proc. of Mass Storage Systems and Technologies. 2010. 1–10.

8 Zaharia M. Job scheduling for multi-user MapReduce clusters. Tech Rep UCB /EECS-2009-55. Berkeley: EECS Department, University of California, 2009.

9 Matei Z, Dhruba B, Joydeep S, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. Proc. of the 5th European Conference on Computer Systems. 2010. 265–278.

10 Hammoud M, Sakr F. Locality-aware reduce task scheduling for MapReduce. Proc. of the 3rd IEEE International Conference on Cloud Computing Technology and Science. 2011. 570–576.

11 Park J. Locality-aware dynamic VM reconfiguration on MapReduce clouds. Proc. of the 21st International Symposium on High-Performance Parallel and Distributed Computing. 2012. 27–36.

12 Zhang XH, Zhong ZY, Feng SZ. Improving data locality of MapReduce by scheduling in homogeneous computing environments. Proc. of the 9th IEEE International Symposium on Parallel and Distributed Processing with Applications. 2011. 120–126.

13 Jin JH, Luo JZ, Song AB, et al. BAR: An efficient data locality driven task scheduling algorithm for cloud computing. Proc. of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2011. 295–304.

14 Guo ZH, Fox G, Zhou M. Investigation of data locality and fairness in MapReduce. Proc of the 3rd International Workshop on MapReduce and its Applications Date. 2012: 25–32.

15 White T. Hadoop: The Definitive Guide. O'Reilly Media, Inc, 2009.

16 Max-min fairness. http://en.wikipedia.org/wiki/Max-min_fairness.

17 Bu X, Rao J, Xu C. Interference and locality-aware task scheduling for mapreduce applications in virtual clusters. Proc. of the 22nd International Symposium on High-Performance Parallel and Distributed Computing. ACM, 2013. 227–238.