Hadoop 集群环境下集成抢占式调度策略的本地性调度算法设计

王越峰 王溪波
(沈阳工业大学信息科学与工程学院 沈阳 110870)

摘 要 在 Hadoop 集群环境下本地性调度算法是提高数据本地性的算法。本地性调度算法的调度策略的本质是提高数据本地性,减少网络传输开销,避免阻塞。但是由于 Map 任务的完成时间不同,Reduce 任务存在的等待现象影响了作业的平均完成时间,使得作业的完成时间增加,进而引起系统的性能参数不佳。因此提出在保留原算法数据本地性要求的基础上集成可抢占式的调度方法。在 Reduce 任务等待时,挂起该任务并释放资源给其他 Map 任务,当 Map 任务完成到一定程度后,重新调度 Reduce 任务。基于上述调度策略设计了集成抢占式策略的本地性调度。为了对改进的算法进行验证,通过实验对本地性调度算法和集成抢占式本地性调度算法进行比较。实验结果表明,在相同数据上,集成抢占式本地性调度算法的平均完成时间有明显的降低。

关键词 数据本地性,抢占式,作业平均完成时间
中图法分类号 TP316.4 **文献标识码** A

Design of Local Scheduling Algorithm for Integrated Preemptive Scheduling Policy in Hadoop Cluster Environment

WANG Yue-feng WANG Xi-bo
(School of Information Science and Engineering, Shenyang University of Technology, Shenyang 110870, China)

Abstract Local scheduling algorithm is an algorithm to improve data locality in Hadoop cluster environment. The nature of the scheduling strategy of the local scheduling algorithm is to improve the data locality, reduce network transmission and avoid congestion. However, due to the different completion time of the Map task, the waiting phenomenon of Reduce task affects the completion average time of the job, the completion time of the job is increased, and then the performance parameters of the system are not good. In this thesis, we proposed to integrate the preemptive scheduling based on the local requirement of the original algorithm. When the Reduce task waits, the task is supended and the resource is rleased to other Map tasks. Based on the above scheduling strategy, this thesis designed the qualitative scheduling of integrated preemptive strategy. In order to validate the improved algorithm, the local scheduling algorithm and the integrated preemptive local scheduling algorithm were compared by experiments. Experimental results show that, on the same data, the average completion time of the integrated preemptive local scheduling algorithm is significantly reduced.

Keywords Data locality, Preemptive, Average completion time for the job

1 引言

Hadoop[1]平台是 Apache 基金组织开发的云计算开源平台,其主要由 Hadoop 提出的 Hadoop 分布式文件系统 HDFS (Hadoop Distributed Files System)[2]和分布式计算框架 MapReduce[3]计算架构组成。

随着 Hadoop 平台的迅速发展,产生了大量数据中心聚集的现象,数据中心的作业处理量也日渐增长。作业调度性能是云计算平台的重要性能指标,良好的作业调度可以缩短作业提交时间并提高准确度,同时也可以充分利用计算资源[4]。Hadoop 中的作业调度是由作业调度器(Task Schedule)实现的,如何设计好的作业调度器对 Hadoop 平台的性能提升尤为重要[5]。

在 Hadoop 中,MapReduce 原有 3 种调度器[6]:1)默认的调度器 FIFO Scheduler(先入先出调度);2)计算能力调度器(Capacity Scheduler);3)公平调度器(Fair Scheduler)。这 3 种 Hadoop 原有的调度器的缺点都非常明显[7]:首先 FIFO 调度器对所有作业都一致,未考虑作业的紧迫程度,因而对小作业的运行不利[8];使用 Capacity Scheduler 时用户需要了解大量系统信息,才能设置和选择队列;Fair Scheduler 不考虑节点的实际负载状态,导致节点负载不均匀。因此越来越多的研究者从多个方面对调度算法进行了深入研究。为了解决资源调度问题,研究者从各个方面做了大量的研究工作[9]。从这些工作中可以总结 4 个热点问题:1)本地性感知任务调度;2)可靠性感知任务调度;3)能量感知资源调度;4)工作流调度。

MapReduce 作业调度算法对集群的性能有着至关重要的影响。Hadoop 系统的性能主要通过以下 5 个指标来衡量[10]:1)平均完成时间,指所有作业的平均完成时间;2)公平性,指调度算法给作业分配资源的公平性;3)数据本地性,指

本文受辽宁"百千万人才工程"培养经费(2012921041)资助。
王越峰(1990-),男,硕士生,主要研究方向为分布式操作系统,E-mail:459617549@qq.com;王溪波(1964-),男,博士,教授,主要研究方向为计算机检测与控制、管理信息系统设计、实时系统及嵌入式软件。

任务在拥有所需数据的节点上执行的比例；4）调度时间，指调度作业的开销；5）调度算法能否满足用户对资源的最小配额要求。然而，这些性能指标之间又是相互矛盾的，即提高某些性能指标必然以其他方面性能的降低为代价。通常，作业调度算法的目标是提高某一项或者几项性能指标。一般来说，作业的平均完成时间是每个调度算法都必须要考虑的性能指标。

2 本地性调度算法

数据本地性是分布式集群环境下作业调度器需要考虑的一个重要指标。由于数据在网络中的传输会产生延迟，尤其在多个机架之间传输时的延迟可能更大，这会导致作业的执行周期变长，而且还会产生大量的网络传输开销。Palanisamy 等人提出 Purlieus[11]，通过将任务调度和数据放置策略相结合的方式来获得更好的 Reduce 任务本地性。他指出如果不考虑数据的放置策略，将很难获得良好的本地性，因为随机的数据放置策略可能会导致一些节点变得更加拥塞。一个有效的数据放置策略需要考虑这些特点，尽量将长作业的数据放到负载最小的节点上。但是这种算法仍然没有考虑到 Reduce 任务的本地性要求。

Hammoud 等人[12]提出本地化感知的 Reduce 任务调度算法 LARTS（Locality-Aware Reduce Task Scheduling for MapReduce）来解决 Reduce 任务数据本地性的问题。LARTS 在 Map 任务完成到一定的阈值 α 后启动 Early Shuffle 机制。这种调度策略利用 Early Shuffle 的优点并且兼顾了 Reduce 任务的数据本地性。但是阈值 α 的设定需要根据不同类型的作业设定，而且存在一定的误差。

3 集成抢占式的本地性调度算法

对于本地性调度算法来说，优先强调的是数据的本地性，但是在满足本地性的同时也要考虑作业调度算法的最重要的指标，即平均完成时间。上文提到 LARTS 本地性调度算法启用了 Early Shuffle 机制，这样就使得 Reduce 任务在 Map 任务还没有全部完成时就开始 Copy Map 任务产生的中间结果。在 Copy 完上一个 Map 任务的中间值后，有可能下一个 Map 任务还没有完成，这会导致 Reduce 任务在占用系统资源的同时等待 Map 任务。这样就浪费了系统的资源，增加了作业的平均完成时间。Reduce 任务与 Map 任务的执行等待关系如图 1 所示。

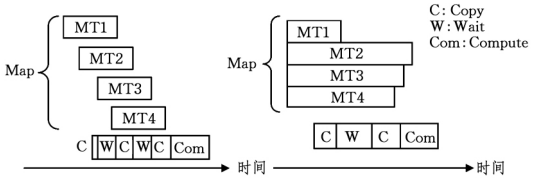


图 1 Reduce 阶段对 Map 阶段的数据依赖关系

Reduce 任务存在空闲等待状态，导致平台资源利用率降低，从而降低了作业执行效率。为了解决资源利用率低和平均完成时间高的问题，本文在原算法的基础上集成可抢占式的作业调度方法。在 Reduce 任务处于空闲等待状态期间，通过抢占其占有的计算资源，将计算资源优化分配给待调度的 Map 任务，从而提高作业执行效率，降低作业的平均完成时间。

MapReduce 的工作可以分成 Map 阶段和 Reduce 阶段。Map 函数处理输入数据，处理结果是一组键值对（key/value）。随后 Reduce 函数处理 Map 函数的输出结果，将具有相同键值的中间结果合并。

Reduce 任务处于空闲等待状态时，抢占问题需要考虑两个关键因素：Reduce 任务占有的计算资源被抢占的时机和 Reduce 任务恢复执行时机。

3.1 Reduce 任务资源被抢占时机

对于 Reduce 任务资源被抢占时机的确定，需要了解 MapReduce 计算框架的基本过程。首先，用户程序把输入数据分割成 M 份，每份为固定大小的数据块（这是 HDFS 存储过程）；然后，开始在集群上进行程序的拷贝，这些程序拷贝中有一份是 Master，其余都是向 Master 请求任务的 Worker。一旦分配到 Map 任务，Worker 便从相应的输入数据中分析出 key/value 对，并把每个 key/value 对作为用户定义的 Map 函数的输入。Map 函数产生的中间值 key/value 对被存储在内存中。存储在内存中的中间值 key/value 对会被定期写入本地磁盘中，用户定义的 Partition 函数将其划分为多个部分，Master 负责把这些中间值 key/value 对在本地磁盘的存储位置传送给执行 Reduce 任务的 Worker。通过远程过程调用，执行 Reduce 任务的 Worker 从执行 Map 任务的 Worker 的本地磁盘读取中间值 key/value 对。当一个执行 Reduce 任务的 Worker 从远程读取到所有所需的中间值 key/value 对之后，通过排序将具有相同 key 的中间值 key/value 对聚合在一起，形成 key/values 对，作为 Reduce 函数的输入。

通过这个过程可以看出，Reduce 任务空闲等待的时间就是在 Reduce 任务拉取中间值和下一个 Map 任务结束之间的时间段。因此要确定 Reduce 任务资源被抢占时机，则须从这两个方向进行计算。

3.1.1 Map 任务剩余执行时间

Map 任务剩余执行时间是指从当前时刻到 Map 任务执行结束时刻的时间段长度，Map 任务剩余时间的计算公式为：

$$T_l=\frac{1-P}{P}\times T \tag{1}$$

其中，T_i 是要得到的 Map 任务剩余执行时间；T 指 Map 任务已经执行的时间；P 是指 Map 任务的执行进度，是通过用已完成的大小除以总任务的大小得到的。

在调度算法中添加 MapRemainingTime 模块，使该模块按照式（1）估计 Map 任务的剩余执行时间，MapRemainingTime 模块主要包含 3 个步骤：1）获取 Map 任务已执行时间与当前的执行进度；2）通过公式计算剩余执行时间；3）通过 TaskReporter 组件将得到的估计值周期性地发送给 ApplicationMaster 组件。通过这个过程估计出 Map 任务剩余执行时间。算法描述如下：

```
Public void MapRemainingTime(Context conf,int m_id,int val)
{ int m_exe_time=get_maps_time(conf,m_id);
//获取 Map 任务执行进度
int m_progress=get_maps_progress(conf,m_id);
int m_re_time=get_maps_remain_time(conf,m_exe_time,m_progress);
send_m_to_AM(m_re_time);
}
```

3.1.2 Reduce 任务拷贝剩余时间

Reduce 任务拷贝剩余时间是指从当前时刻到将 Map 任务完成后的中间值拷贝到 Reduce 任务缓冲区的时间段长度，

计算公式为：

$$T_l=(T_m+nT_b)(1-P) \tag{2}$$

其中，T_l 是 Reduce 任务拷贝剩余时间；T_m 是拷贝 Map 任务中间值的时间；T_b 是不同机架间的传输时间。P 是指 Reduce 任务拷贝进度。

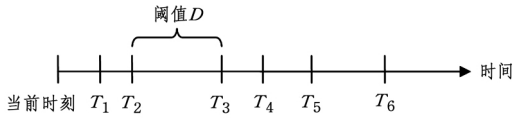
在调度算法中添加 ReduceRemainingTime 模块，使该模块按照式(2)估计 Reduce 任务的拷贝剩余时间，ReduceRemainingTime 模块主要包含 3 个步骤：1)获取拷贝 Map 任务时间、机架间传输时间与拷贝进度；2)通过公式计算剩余拷贝时间；3)通过 TaskReporter 组件将得到的估计值周期性地发送给 ApplicationMaster 组件。通过这个过程估计出 Reduce 任务拷贝剩余时间，算法描述如下：

```
Public void ReduceRemainingTime(Context conf,int r_id,int m_id,int
val)
{ int m_time=get_maps_time(conf,m_id);
  int mr_tr_time=get_mr_tr_time(conf,m_id,r_id);
  int mr_cp_prgress=get_mr_cp_progress (conf,m_id,r_id);
//通过公式计算剩余拷贝时间
int mr_cp_re_time=get_mr_cp_re_time (conf,m_time,mr_tr_time,
mr_cp_progres)
Send_r_to_AM(mr_cp_re_time);
}
```

根据 Map 任务的剩余执行时间的估计值和 Reduce 任务拷贝剩余处理时间的估计值估计 Reduce 任务资源的被抢占时机，由估计结果判断是否需要挂起 Reduce 任务，如式(3)所示：

$$\mathrm{Min}(T_{l\_map1},T_{l\_map2},\cdots,T_{l\_mapn})-\mathrm{Max}(T_{l\_reduce1},T_{l\_reduce2},\cdots,T_{l\_reducen})\geqslant D \tag{3}$$

将正在执行的 Map 任务对应的剩余执行时间的估计值和 Reduce 任务拷贝剩余处理时间的估计值作为评估 Reduce 任务是否挂起的输入参数，参数值由 ApplicationMaster 组件负责收集。ApplicationMaster 组件接收到周期性的参数后，Reduce 挂起模块通过两个步骤判断是否挂起 Reduce 任务：1)计算最小Map 任务剩余执行时间与最大 Reduce 任务拷贝时间的差值；2)比较差值与阈值的关系以确定是否抢占。若差值大于设定阈值，在 TaskManager 组件中将 Reduce 任务的状态设置为等待挂起状态，如图 2 所示。



T_1：Reduce任务1拷贝剩余处理时间估计值
T_2：Reduce任务2拷贝剩余处理时间估计值
T_3：Map任务1剩余执行结束时间估计值
T_4：Map任务2剩余执行结束时间估计值
T_5：Map任务3剩余执行结束时间估计值
T_6：Map任务4剩余执行结束时间估计值

图 2　任务执行时间图

算法描述如下。

```
Public void Cal_MrAndRr_diffValue(Context conf,job_id)
{ int min_m_re_time=get_Min_MapRemaining(conf,job_id);
  //获取最大任务拷贝时间的 Reduce 任务
  int max_mr_cp_progress=get_Max_ReduceRemaining (conf,job_
  id);
  //比较差值与阈值
  boolean result=get_MR_compare(conf,min_m_re_time,max_mr_
  cp_progress);
if(resule){set_Reduces_hang(conf,job_id);}
else{set_Reduces_running(conf,job_id);}
}
```

3.2　Reduce 任务重调度时机

Reduce 任务被抢占后释放占用的资源，资源被分给有需要的 Map 任务。但是此时 Reduce 任务还处于挂起状态，何时结束挂起状态是需要研究的内容。当部分 Map 任务执行结束时，Reduce 任务需要从这些新产生的执行结束的 Map 任务中拉取中间值，这就是 Reduce 任务从挂起状态恢复执行的触发条件。Reduce 任务重调度模型即是根据 Reduce 任务挂起后新产生的执行结束的 Map 任务个数判断是否需要将 Reduce 任务从挂起状态恢复，从而继续执行。为避免由 Reduce 任务频繁执行挂起、恢复操作而增加的系统调度时间，当 Reduce 任务挂起后执行结束的 Map 任务个数达到设定阈值时，才将 Reduce 任务从挂起状态恢复执行，如式(4)所示：

$$\frac{N_s-N_f}{N_s}\geqslant D \tag{4}$$

其中，N_s 表示已完成的 Map 任务个数，N_f 表示已经成功拷贝到 Reduce 缓冲区的中间值个数。Reduce 重调度模块通过两个步骤判断是否恢复 Reduce 任务：1)计算在抢占过程中完成的 Map 个数所占的比值；2)比较计算结果与设定阈值，判断是否该重调度。若差值大于设定阈值，在 TaskManager 组件中将 Reduce 任务的状态设置为运行状态。算法描述如下：

```
Public void Restart_diapatch_Reduce(Context conf,int job_id)
  { //获取 Map Tasks 执行完成数
  int map_finish_nums=get_finish_map_task(conf,job_id);
  //比较差值与阈值
  Boolean result=get_MR_compare(conf,map_finish _nums);
  //true 代表差值大于等于设定阈值
  //false 代表差值小于设定阈值
  if(result){
  //设定 Reduce Task 挂起
  Set_Reduces_running(conf,job_id);}
  else{ set_Reduces_hang(conf,job_id);}
  }
```

4　实验结果及性能分析

本文通过虚拟机的方式搭建异构测试环境。定义两个机架，每个机架有 5 台虚拟机，每个虚拟机分配 1GB 内存。测试作业为 WordCount。测试集成抢占式的本地性作业调度和原算法的 Reduce 任务的平均等待时间。结果如图 3 所示，集成可抢占式的调度算法后 Reduce 任务的平均等待时间平均降低 85.23%，最高降低 87.69%，说明了集成抢占式的本地性调度的 Reduce 任务的平均等待时间得到大幅度降低。

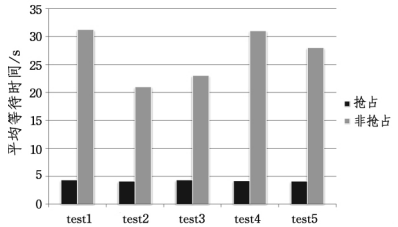


图 3　Reduce 任务平均等待时间的比较

为对集成抢占式策略的本地性调度算法进行平均完成时间的测试，测试文本大小为 4GB，结果如图 4 所示。本实验对集成抢占式本地性调度算法、非抢占本地性调度算法和集成抢占式非本地性调度算法进行测试。通过观察发现，集成抢占式本地性调度的 Reduce 任务平均完成时间较非抢占式本

地性调度平均降低 14.12%，最高降低 17.01%，说明了集成抢占式的本地性调度算法的平均完成时间得到降低。同时比较集成抢占式非本地性调度算法可以发现，其完成时间也有降低，说明这种算法在降低完成时间的同时，保证了数据本地性。

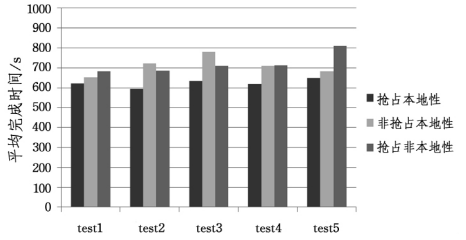


图 4 平均完成时间的比较

接下来测试不同的 Map 任务个数对于两种算法的影响，结果如图 5 所示。从集成了抢占式的调度算法后与非抢占式的本地性调度算法的对比可以看出，当 Map 任务个数较少时，其并没有明显的差别；而伴随 Map 任务个数的增加，两种算法的平均完成时间差值在逐渐增大。由实验数据可知，对于集成了可抢占式的本地性调度算法，Map 任务数越大，调度算法就越优。

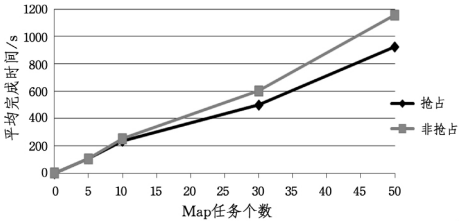


图 5 不同 Map 任务数的平均完成时间比较

结束语 本文分析了 Hadoop 集群下对数据本地性调度的改进，并指出在保证原算法的数据本地性的前提下可以通过抢占的方式减少作业的平均完成时间。通过分析 MapReduce 计算框架，找出抢占与重调度时机，从而达到抢占空闲 Reduce 任务的目的。通过实验可以发现，集成了可抢占式的本地性作业调度的平均等待时间、平均完成时间都有了不同程度的降低。

这种集成了抢占式的本地性调度算法依然存在一些不足，例如添加抢占和重调度机制后增加了系统开销。测试的作业功能和数据类型不全面，在大数据情况下的性能测试还不是很多，实验在普遍性上还有所不足。接下来的工作重点是研究如何降低系统开销以及当开销为何值时可被接受等问题，同时将在更大的实验数据集上进行改进和验证。

参 考 文 献

[1] Hadoop[EB/OL]. [2014-2-01]. http//hadoop. apache. org.

[2] SHVACHKO K, KUANG H, RADIA S, et al. The hadoop distributed file system[C]// Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies. IEEE, 2010: 1-10.

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

[4] ZAHARIA M, BORTHAKU D, SARMA J S, et al. Job Scheduling for Multi-user Mapreduce Clusters[R]. EECS Department, University of California, Berkeley, Tech. 2009.

[5] 董西成. Hadoop 技术内幕：深入解析 MapReduce 架构设计与实现原理[M]. 北京：机械工业出版社，2013.

[6] 胡丹，于炯. Hadoop 平台下改进的 LATE 调度算法[J]. 计算机工程与应用，2014，50(4)：86-89.

[7] 何文峰. 基于任务特征与公平策略的 Hadoop 作业调度算法研究[D]. 武汉：华中科技大学，2013.

[8] 燕明磊. Hadoop 集群中作业调度研究[J]. 软件导刊，2015，14(4)：1-2.

[9] 储雅，马廷淮. 云计算资源调度：策略与算法[J]. 计算机科学，2013，40(11)：8-13.

[10] 陶昌俊. Hadoop 平台的作业调度算法[D]. 合肥：中国科学技术大学，2015.

[11] PALANISAMY B, SINGH A, LIU L, et al. Purlieus: locality-aware resource allocation for MapReduce in a cloud[C]// Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. 2011.

[12] HAMMOUD M, SAKR M F. Locality-Aware Reduce Task Scheduling for MapReduce[C]// Proceedings of International Conference on Cloud Computing Technology & Science. Beijing, 2011: 570-576.

(上接第 559 页)

[10] WAWRYN K. A Formal Language Description and Inference Strategy for Aanalog Circuit Design[J]. Circuits Systems Signal Processing, 1996, 15(6): 771-805.

[11] BUTHAYNA H. Eilouti A Formal Language for Palladian Palazzo Façades Represented by a String Recognition Device[J]. Nexus Network Journal, 2008, 10(2): 245-268.

[12] ALUR R, DILL D. Automata for modeling real-time systems[C]// 17th Internat. Colloquium on Automata, Languages and Programming (ICALP 90). Lecture Notes in Computer Science, Springer, Berlin, 1990.

[13] ZHANG X W, LI Y M. Intuitionistic fuzzy recognizers and intuitionistic fuzzy finite automata[M]. Springer-Verlag, 2009.

[14] KRITHIVASAN K, SHARDA K. Fuzzy w-automato[J]. Information Science, 2001, 138(2001): 257-281.

[15] WANG J, YIN M, GU W. Fuzzy multiset finite automata and their languages[J]. Soft Comput, 2013, 17(3): 381-390.

[16] KRITHIVASAN K, SHARDA K. Fuzzy w-automata[J]. Information Science, 2001(138): 257-281.

[17] BIRKEDAL L, ROTHWELL N, TOFTE M, et al. The ML kit (version 1)[R]. Tech. Report DIKU-Report 93＝14, Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, 1993.

[18] MIKL N A. Real Zeros and Normal Distribution for Statistics on Stirling Permutations Defined by Gessel and Stanley[J]. Siam Journal on Discrete Mathematics, 2007, 23(1): 401-406.

[19] KARNEY C F F. Sampling Exactly from the Normal Distribution[J]. ACM Transactions on Mathematical Software, 2016, 42(1): 3.

[20] ABRIAL J R. The B-Book[M]// Cambridge University Press. Cambridge, 1996.

[21] ALUR R, COURCOUBETIS C, DILL D. Model-checking for real-time systems[C]// IEEE 5th Annual Symp. on Logic In Computer Science. Philadelphia, 1990: 414-425.

[22] ALUR R, DILL D. Automata for modeling real-time systems[C]// 17th Internat. Colloquium on Automata, Languages and Programming (ICALP 90). Lecture Notes in Computer Science, Springer, Berlin, 1990.