# 一种周期性 MapReduce 作业的负载均衡策略

傅 杰 都志辉
(清华大学计算机科学与技术系 北京 100084)

**摘 要** MapReduce 任务负载均衡主要是通过分区函数来实现的,Hadoop 默认的分区函数并不能很好地保证 reducer的负载均衡。针对周期性的业务处理提出了一种基于权重计算的负载均衡策略,周期性任务的数据分布与历史数据相比具有相似性。本策略根据历史数据运行的信息运算出数据权重信息(文中用权重表示每条记录的处理复杂度),再通过 Map 阶段抽样分析当前这批数据的分布特征来预测待处理数据带权重的整体近似分布情况,从而指导 Reduce 分区,以保证其负载均衡。通过简单的例子仿真了整个策略的运作过程,并且对比了与 TeraSort 思路的不同点。最后通过分析用户访问视频的日志证明了文中提到的策略比默认的策略性能提高了接近 1 倍。

**关键词** MapReduce,TeraSort,负载均衡,周期性
**中图法分类号** TP311.1 **文献标识码** A

## Load Balancing Strategy on Periodical MapReduce Job

FU Jie DU Zhi-hui
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

**Abstract** The MapReduce task load balancing in Hadoop mainly depends on the partition function. The Hadoop default partition function is not efficient in practical business processing. This paper presented a load balancing strategy based on the weight value of the periodic jobs. Because the data's distribution is similar in each period, we calculated the weight from historical data's profile. Through analyzing a sample data in Map phase to predict the whole data weighted integral approximate distribution, the strategy guids the Reduce partition to ensure its load balancing. We also presented the difference between TeraSort strategy and the new strategy. The experimental results with the view video logs show that the performance of our strategy is improved about 2 times compared with the default strategy.

**Keywords** MapReduce, TeraSort, Load balance, Periodic

## 1 背景

随着互联网的高速发展,互联网用户每天消耗在互联网的时间越来越多,对于互联网企业来说,每天需要处理的用户日志数以 T 计。面对这种周期性的计算任务,只要提高一些性能,通过时间的积累,就可获得可观的效益,不仅能提高对系统的响应,还可以节约计算资源,降低成本[1]。目前互联网企业大多采用 MapReduce 计算框架来分析用户日志,但是由于业务数据的复杂性,经常会出现同一批业务数据其处理效率不尽相同,加上 Hadoop 自身的架构特点,其在 Reduce 阶段会出现负载不均的情况[2,3]。Hadoop 默认提供了 Hash Partition 负载均衡策略,但其不能很好地解决这类问题,所以研究这种周期性 MapReduce 任务的负载均衡策略是很有现实意义的。

## 2 哈希分区算法

在 MapReduce 中,Map 的输出结果主要是通过 Partition 分发到相应的 Reducer 上,Hadoop 默认提供的是 Hash Partition 算法,其主要思想就是以 Key 的哈希值对 Reducer 进行求模,得到相应的 Reducer 序号,这样能保证具有相同 Key 的数据被分发到一个 Reducer 中进行处理,但并不能保证 Reducer 的负载均衡[4,5]。如果出现某个 Key 的数据量极大,就会导致某个 Reducer 负载过大,从而降低整个作业运行的效率。所以在一般的业务处理中如果遇到了 Reduce 数据倾斜问题,就需要重写 Partition 方法来达到 Reduce 的负载均衡。

## 3 TeraSort 排序

2008 年,Hadoop 利用 Tearsort 算法在 1T 排序评估中获得第一名,耗时 209 秒[6]。其排序是在 Reduce 阶段完成的,在 Map 开始阶段采用抽样数据提取摘要,并对摘要进行运算得出一个分发标记,这样就能比较有效地将 Map 输出结果均等地分发到 Reduce 节点上,此外 Map 与 Reduce 也可以并行,以达到更高的效率。具体步骤如图 1 所示,主要是通过 3 个步骤来实现的,首先是对数据进行抽样,MapReduce 数据的输入都通过 InputFormat 进行;然后切分成各个子 Map Task,所以 TeraSort 是在这个阶段对数据进行抽样的,系统

到稿日期:2012-10-19 返修日期:2013-01-10 本文受国家自然科学基金((61272087,61073008,60773148,60503039),北京市基金(4082016,4122039)资助。

傅 杰(1985—),男,硕士,主要研究方向为海量数据处理、hadoop 生态技术,E-mail:kojie.fu@gmail.com;都志辉 男,博士,副教授,主要研究方向为网格计算、海量数据处理等。

配置了一个抽样参数：terasort. partitions. sample，其默认值为100000条[7]。其次是根据Map函数输出的Key对抽样的数据进行统计排序，将这些Key相对平均地分配到$R$组中，其中$R$为配置的Reduce数量。这个分组过程既要保证相同的Key都在同一个组内，又要保证每个组间的数据也是有序的，然后取每组的边界作为分发标记_partition. lst写入到缓存中[8]。这样，通过_partition. lst得知每个Key应该往哪个Reducer服务器上分发，在匹配_partition. lst的时候是通过TireTree来查找的，所以效率比较高。通过以上3步既能很好地保证MapReduce的并行性，又能保证Reducer的负载均衡[9]。

## 4 基于权重信息的分区负载均衡策略

### 4.1 策略设计

TeraSort算法比较巧妙的地方在于通过一个摘要的提取感知整体数据的分布情况，这种方式对实际业务中处理海量数据会有很大的性能提升[10]。可以看出，TeraSort在处理偶然性或者一次性的MapReduce计算时其设计思路非常巧妙且很有效果，但是对于周期性的MapReduce任务，这种策略并不是最有效的，特别是对于不同的Key，其在Reduce阶段单条记录处理效率不同的情况下，也会出现工作量倾斜。在实际业务场景中，在同一个Job中处理不同Key会出现有些记录的处理复杂度要明显高于另外一些，通过TeraSort的分区机制只能保证Reduce Task中记录数大致均等，反而导致其计算量的倾斜。基于这样的场景，本文提出了一种新的负载均衡策略——基于权重信息分区策略，来保证Reduce Task工作量的均等性而不是单纯数量的均等。具体如图1红色部分所示，整个设计在原来的基础上新增了两个模块，其一，在抽样统计的过程中不再是单纯地按照频率进行统计，而是通过权重值来切分_partition. lst。这个权重信息在提交MapReduce任务的时候通过configuration设置到缓存里面。其二，在Reduce阶段进行的Counter统计，主要是用来统计每个Key处理的平均时间并且算出相对的权重值[11]。

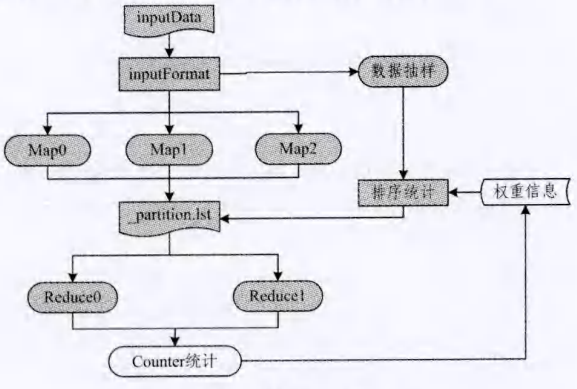


图1 基于权重的partition方案

#### 4.1.1 权重信息模块

本策略设计了一个持久层来存储每个Key对应的权重信息，在提交任务的时候将权重信息通过DistributedCache分发到临时缓存里，其在内存中的存储结构是一个HashMap〈key，value〉，其中key即为Map阶段输出的Key，value则为该Key相应处理的难度权重值。权重值越大，则代表相同的环境下该Key处理所需的时间越长。权重的取值范围为0到无穷大，一般来说集中在1附近，默认取值为1。在进行抽样排序统计时，先通过该HashMap获取到该Key相应的权重，如果没有获取到该权重，则默认取1。下面通过实例加以说明。

表1 抽样Key出现频率信息

| 抽样Key | 出现频率 | 单位权值 | 综合权值 | 分区Tag |
|---|---|---|---|---|
| a | 2000 | 0.1 | 200 | |
| abc | 300 | 1.0 | 300 | |
| Bc | 200 | 2.0 | 400 | |
| bd | 400 | 0.5 | 200 | Tag0 |
| cf | 1000 | 0.15 | 150 | |
| dg | 1000 | 1.0 | 1000 | |
| eq | 400 | 2.0 | 800 | Tag1 |
| gk | 200 | 1.0 | 200 | |
| wd | 100 | 1.5 | 150 | |

假设Key对应的权重值如表1第二行所示(其根据历史记录运算出来的)，且Reduce数为3，则先根据抽样运算出每个Key的出现频率，如表1第一行所示，加载每个Key相应的单位权值，如第二行所示，再将频率与单位权值相乘算出综合权值，如第三行所示。最后将综合权值数组进行三等分，算出两个分割点，如表1所列，分割点为bd、eq，其中综合权值＝出现频率＊单位权值；当在Map中未找到相应的Key权重时，默认取1。如上表abc对应的单位权值取1。若是按照Tearsort的方式，所得到的分割点将会是a、cf。分区不同将直接影响reduce的负载。

#### 4.1.2 Counter统计模块

Counter统计实际上是对权重信息的一个反馈，在实际的Reduce运行过程中监控每个Key的运行效率，统计每个Key运行的实际消耗时间以及平均消耗的时间。在权重信息数据中保存了所有的Key运行的历史总次数以及总时长，每次Reduce运行完以后都会将该数据增量地添加到权重信息中，同时对整个权重信息进行平衡调整。取所有Key的平均消耗时间的权重为1，其他Key的平均消耗时间相对于此求出相对的权重，并将该权重值更新到权重信息文件中。这样随着任务的不断运行，该权重信息将不断地训练、调整，以保证Reducer的负载均衡。

## 5 实验对比

实际的工程应用中会经常出现一批数据存在多种类别，其处理流程一样，但是处理的复杂度不相同。比如我们有一类任务主要是分析客户访问视频网站的日志，该日志包含了目前互联网视频网站上大部分视频播放的URL，我们利用MapReduce来分析用户访问过的视频所属的题材信息。因为每类网站其分析流程虽然一样，但是涉及到的转换逻辑存在复杂与简单之分。下面主要针对这样的一个场景来比较Hadoop默认Partition方案与本策略的效率差别。

### 5.1 实验环境

实际工程应用的整个实验都是在Hadoop平台下运行的，采用最新1.0.3的版本，硬件设备由3台机器(CPU：Intel Xeon E5506，内存为4G)组成，分别为nutch、data1、data2，其中nutch既为NameNode，又为JobTracker，其他节点为DataNode。

### 5.2 实验结果

考虑到实验的公平性，采用同一批数据、同一个网络环

境、同一个实验平台进行这两个实验，两个 Job 的 Reduce 数量都设置为 4。默认 Partition 方案将不做任务处理而直接进行该实验。本方案第一次运行不带任何权重信息，实际上就是 TeraSort 所采用的方案。所以我们实际上是先随意地运行一些其他数据生成一个初始的权重信息，再利用给定的实验数据进行加权方案的测试。实验结果如表 2 和表 3 所列。

表 2　默认 hadoop 实验结果数据(2012. 6. 18)

| Task | 开始时间 | 结束时间 | 消耗时间 |
|---|---|---|---|
| Task021_00 | 12:48:45 | 13:48:37 | 59mins,52sec |
| Task021_01 | 13:01:45 | 14:07:34 | 65mins,49sec |
| Task021_02 | 13:07:45 | 14:21:37 | 73min,52sec |
| Task021_03 | 12:58:45 | 15:15:36 | 196min,50sec |
| 总时间 | 12:48:35 | 15:15:51 | 207mins,16sec |

表 3　本方案实验结果数据(2012. 6. 18)

| Task | 开始时间 | 结束时间 | 消耗时间 |
|---|---|---|---|
| task_r_022_00 | 15:50:07 | 16:03:30 | 73mins,23sec |
| task_r_022_01 | 15:55:07 | 16:15:06 | 89mins,59sec |
| task_r_022_02 | 15:53:07 | 16:20:15 | 87mins,7sec |
| task_r_022_03 | 15:59:07 | 16:28:42 | 89mins34sec |
| 总时间 | 15:49:57 | 16:29:00 | 99mins,3sec |

从图 2 可以看出，优化前后效率提升接近一倍。所以对于同一个 Job 中存在处理复杂度不一样的 Key，采用文中给出的 Partition 方案，效率会有明显的提升，处理的复杂度差别越大，则效果就越明显。通过对上表两种实验场景各个 Reducer 运行的时间求标准差可明显看出，采用本方案以后负载比默认方案要均衡，整个 Job 效率也因为 Reduce 阶段的负载均衡而得到明显提升。

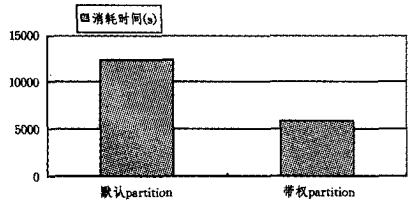


图 2　实验消耗时间对比图

**结束语**　在实际的业务中，周期性的海量数据计算是非常普遍的，而且周期性的数据在一定时间内其数据分布具有相似性。本文设计的策略充分利用周期性任务的这个特点，通过历史的统计运行信息来预测将来数据的分布，后续可以在历史数据基础上设计出一些更为精准的预测算法来提升负载的均衡度。本策略具有可调性、记忆性以及沉淀性，运行的时间越长，其统计出来的每种类型的数据的权重信息越丰富、越准确。

## 参 考 文 献


[1] White T. Hadoop: The definitive guide [OL]. http://books.google.com,2010

[2] Borthakur D. TheHadoop Distributed File System: Architecture and Design[OL]. http://cloudcomputing.googlecode.com,2007

[3] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[C]//OSDI'04, Proceedings of the 6th Coference on Symposium Opearting Systems Design & Implementation. Sep. 2004

[4] Lammel M R. Google's MapReduce programming model—Revisited[J]. Data Programmability Team,2007,68(3):208-237

[5] Armbrust M, Fox A, Griffith R. Above the Clouds: A Berkeley View of Cloud Computing[M]. ACM,2010

[6] Seo S, et al. HPMR: Prefetching and Pre-shuffling SharedMapReduce Computation Environment[C]//the Proceedings of 11th IEEEInternational Conference on Cluster Computing. Sep. 2009

[7] Jiang D,Ooi B C,Shi L,et al. The Performance of MapReduce: An Indepth Study[C]//Int'l Conference on Very Large Data Bases (VLDB). 2010

[8] Dittrich J,Jindal A. Schad Hadoop++:Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)[J]. VLDB 2010/PVLDB,2010,34(1/2):515-529

[9] Liu Xu-hui,Han Ji-zhong. Implementing WebGIS on Hadoop:A case study of improving small file I/O performance on HDFS[C]// Cluster Computing and Workshops, 2009. IEEE International Conference on. 2009:1-8

[10] Lee K-H,Lee Y-J,Choi H,et al. Parallel data processing with MapReduce:a survey[J]. ACM SIGMOD Record,2011,40(4):11-20

[11] O'Malley O. TeraByte Sort on Apache Hadoop[Z]. Yahoo!, May 2008


(上接第 23 页)


[39] Foster K A. Sensitive test data for logic expressions [J]. ACM SIGSOFT Software Engineering Notes,1984,9(2):120-125

[40] Yu L, Tsai W T, Zhao W, et al. Towards Selecting Test Data Using Topological Structure of Boolean Expression [C]// Proceedings of 9th International Conference on Quality Software (QSIC 2009). IEEE Computer Society,2009:31-40

[41] Chen T Y, Lau M F. An Empirical Study on the Effectiveness of the Greedy MUTP Strategy [C]// Proceedings of International Conference on Software Engineering: Education and Practice (SEEP'98). California: IEEE Computer Society,1998:338-344

[42] Chen T Y,Grant D D,Lau M F,et al. BEAT:Boolean Expression Fault-based Test Case Generator [C]// Proceedings of International Conference on Information Technology: Research and Education (ITRE 2003). California: IEEE Computer Society, 2003:64-69

[43] Sun C A,Sim K Y,Tse T H,et al. An Empirical Evaluation and Analysis of the Fault-Detection Capability of MUMCUT for General Boolean Expressions [C]//Proceedings of International Computer Symposium (ICS 2004). Taipei,2004:926-932

[44] Chen T Y, Lau M F. Test case selection strategies based on Boolean specifications [J]. Software Testing, Verification and Reliability,2001,11(3):165-180

[45] Sun C A,Sim K Y. An FSM-based Parameterized Generator of General Boolean Expressions [C]//Proceedings of International Computer Engineering Conference (ICENCO). 2004:119-126