文章编号:1000-5862(2013)01-0051-05

一种基于处理时间的 Map/Reduce 动态均衡调度算法

陈 军[1,2],卢涵宇[1,3],姚丹丹[1]

(1. 成都理工大学地球探测与信息技术教育部重点实验室,四川 成都 610059;
2. 成都信息工程学院资源环境学院,四川 成都 610225;3. 贵州大学计算机与信息工程系,贵州 贵阳 550003)

摘要:提出了一种基于处理时间的 Map/Reduce 动态均衡调度算法. 在该算法中,为参与计算的各节点建立处理总时间指标;当节点的子任务返回时动态更新当前节点的处理时间,并按照最小处理总时间来选择子任务处理的节点. 实验证明:该算法不仅减少了节点与云中心服务器的负载信息传递,而且充分利用了各计算节点的处理能力,提高了 Map/Reduce 调度效率,对于提升云计算服务的质量具有十分重要的理论意义.

关键词:云计算;映射/规约;负载均衡调度

中图分类号:TP 391 **文献标志码**:A

0 引言

Map/Reduce 是 Google 提出的一个软件架构,用于大规模数据集(大于 1 TB)的并行运算. 正是通过 Map/Reduce 技术,Google 每天能处理超过 10 亿条来自世界各地的查询,而平均回复搜寻结果时间为0.25 s[1]. Map/Reduce 隐藏了并行化、容错、负载均衡等技术细节,为并行系统的数据处理提供了简单的解决方案. 作为一种并行计算技术,Map/Reduce 具有分布式海量数据处理的能力,是云计算的核心技术之一. 在云平台上,Map/Reduce 被划分为大量的子任务提交至计算节点完成. 如果没有合理的调度机制,在某一时刻,某些计算节点承担的任务很重,而其它的计算节点则很空闲,使 Map/Reduce 计算效率降低. 因此,研究合理的计算调度机制对于提高云计算的效率和质量至关重要[2]. 本文提出一种基于处理时间的 Map/Reduce 动态均衡调度调度算法,实验证明,该算法能有效地进行 Map/Reduce 任务调度,具有一定的理论意义和实用价值.

1 Map/Reduce 的基本原理

在 Map/Reduce 应用于云计算之前,程序员为了处理海量的原始数据,使用了诸如文档抓取(类似网络爬虫的程序)、Web 请求日志等各种计算方法. 大多数这样的数据处理运算在概念上很容易理解. 然而由于输入的数据量巨大,要想在可接受的时间内完成运算,只有将这些计算分布在成百上千的主机上. 如何处理并行计算、如何分发数据、如何处理错误都需要通过代码实现,大量的代码使得原本简单的运算变得难以处理.

为了解决上述复杂的问题,设计师们提出了一个抽象模型. 在该模型中,只要表述想要执行的简单运算即可,而不必关心并行计算、容错、数据分布、负载均衡等复杂的细节,这些问题都被封装在了一个库里面. 设计这个抽象模型的灵感来自 Lisp 和许多其它函数式语言的 Map 和 Reduce 的原语,也就是后来的云计算核心技术 Map/Reduce[3-4].

Map/Reduce 即映射(Map)和规约(Reduce). 如图 1 所示,首先将计算任务进行切割,分成多个独立的、可并行的子任务(Split). 这些子任务通过网络传送至计算节点上执行 Map 操作. Map 操作输入参数被抽象为一个键值对(Key-Value). Map 操作的结果生成中间状态的键值对(Key-Value). 一个 Reduce 操作负责将相同键值的结果进行合并,生成最终的结果集[5-6].

在 Map/Reduce 编程模型中,任务的分发、节点间的负载均衡调度以及容错机制等被封装在开发库

收稿日期:2012-10-11
基金项目:国家自然科学基金(61071121)资助项目.
作者简介:陈 军(1979-),男,四川南充人,讲师,博士研究生 主要从事遥感与 GIS 技术的研究.

中,程序员只需实现 Map 和 Reduce 函数,从而使复杂的问题被大大简化.

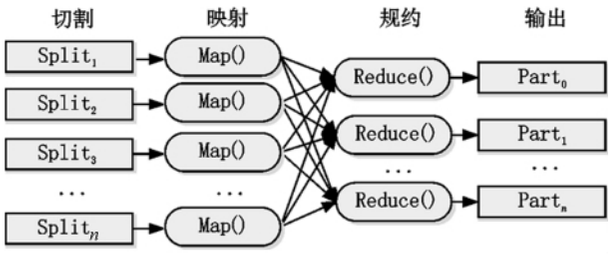

图 1 Map/Reduce 的编程模型

2 分布式系统负载均衡调度主要算法

Map/Reduce 负载均衡调度属于分布式系统的节点调度问题. 在分布式系统中,任务调度由 4 部分构成[7-9]:(ⅰ)信息采集,对节点当前负载信息的采集;(ⅱ)选择,即任务分配,任务由哪一个计算节点完成;(ⅲ)均衡发起,任务调度的调用方,可以是节点,也可以是任务调度的服务器;(ⅳ)协商,计算节点间通过协商决定负载迁移的问题.

这 4 个部分可总结为 4 个关键问题[10-11],即负载状况的定义、负载的分配、负载探测与均衡策略. 在这 4 个问题中,以均衡策略为最关键,决定了整个分布式系统的任务执行效率、响应能力和吞吐量. 目前,负载均衡调度策略分为资源负载均衡策略、发送者启动策略和接受者启动策略等[11].

资源负载均衡策略又分为动态局部物理分布策略和全局动态负载均衡策略. 动态局部物理分布策略由 Bryant 等提出,采用启发式方法达到系统的负载均衡. 但该方法中每一个节点均需要遍历其它节点,当网络中节点规模巨大时,消息请求的复杂度非常大,复杂度为 $O(n^2)$,大量的网络内部通讯降低了分布式系统的任务处理效率. 全局动态负载均衡策略由 K. Barak 等[7] 提出,该策略属于协调算法,每一个节点均采集其它节点的资源负载信息,并随时更新. 当某一个节点的负载量超过阈值时,通过该节点存储的其它节点负载信息,将任务迁移至估计响应最小时间的节点来处理. 由于节点通过全部节点的负载状态建立迁移策略,方法可以达到全局最优,但通信量仍然很大,尤其是节点采集的信息不及时,该策略无效.

发送者启动策略最早由 D. L. Eager 等[12] 提出. 在该策略中,任务调度由任务调度的服务器发出. 服务器通过采集节点的负载状态,将任务分配给负载量最小的那个节点. 在该模型中,服务器需要随时采集节点的负载信息,增加了节点与服务器之间的网络通信次数,尤其对于正处于繁忙状态的节点,服务器对节点状态的查询无疑增加了节点的负担.

接收者启动策略由 M. Livny 等[13] 提出,在该策略中,任务请求的发起者为空闲节点. 节点通过对本机的负载量评价,当负载量小于阈值时为空闲节点. 空闲节点向邻近节点发送任务请求,若邻近节点没有满足要求的请求,该节点继续等待. 由于接收者策略采用自评价机制,避免了负载信息的网络交换. 但是,当网络中存在大量空闲节点时,任务请求对于繁忙节点是极大的干扰,从而影响整个系统的运行效率.

在以上策略中,以资源动态负载均衡最简单,但系统内部通信量大,计算复杂度高; 发送者启动策略适合于系统整体负载较低的情况; 接收者启动策略则适合于系统整体负载较高的情况,在大规模的分布式并行处理中得到广泛运用. 如 Hadoop 的均衡调度,就使用类似算法实现[14].

3 基于处理时间的计算节点动态均衡调度

针对前述几种调度算法的主要问题,采用一种基于处理时间的动态均衡调度策略对计算节点进行负载均衡. 该策略分为两部分: 负载信息采集与均衡策略.

3.1 负载信息采集

负载信息采集以云中心服务器为容器,采集计算节点的负载信息. 常用负载指标有 CPU 利用率、内存使用情况、网络流量、进程响应时间等[15]. 本文采用计算节点为发起源的策略采集 CPU 状态、可用内存、可用磁盘空间、平均流入量、平均流出量、节点最大响应时间等指标. 计算节点定时向中心服务器发送表示节点状态的数据包,服务器接收后,读出节点状态信息并保存.

当客户端请求计算任务时,将节点最大响应时间作为最高优先级指标. 指标的选取以及优先级如表 1 所示.

表 1 计算节点负载均衡指标与优先级

指标	最大响应时间	CPU 利用率	可用内存	网络平均流量
优先级	1	2	3	4

在负载均衡调度中,按表 1 所示的优先级从中心服务器获得有序的节点列表. 第 1 次调度按照节点列表定义的顺序依次向相应的计算节点提交 Map 请求.

3.2　基于处理时间的动态均衡策略

节点的负载信息处于不断变化中. 当节点处于繁忙状态时,向中心服务器提交负载信息可能不及时. 因此,依靠中心服务器返回的节点列表不能完全信任,需要建立动态的均衡调度策略.

在任务分配时,为每一个节点建立一个动态的负载评估机制. 子任务在节点执行的过程,也是负载量动态评估的过程.

设有 n 个计算节点参与计算,任务总量为 1,节点的平均处理速度和运行时间分别为 V_i 和 T_i,建立如下方程:

$$\sum_{i=1}^{n} V_i \times T_i = 1. \tag{1}$$

考察第 1 个计算节点 N_1,将(1)式变形为

$$V_1 \times T_1 = 1 - \sum_{i=2}^{n} V_i \times T_i. \tag{2}$$

设其余计算节点按内部最佳调度策略调度,那么对于节点序列 $N_j(2 \leqslant j \leqslant n)$,存在一个处理时间最大的节点 p_1,满足

$$T_{p_1} = \max(T_2, T_3, \cdots, T_n), \tag{3}$$

理论上,T_1 与 T_{p_1} 存在函数相关关系为

$$T_{p_1} = f(T_1). \tag{4}$$

该函数为单调递减函数,如图 2 所示.

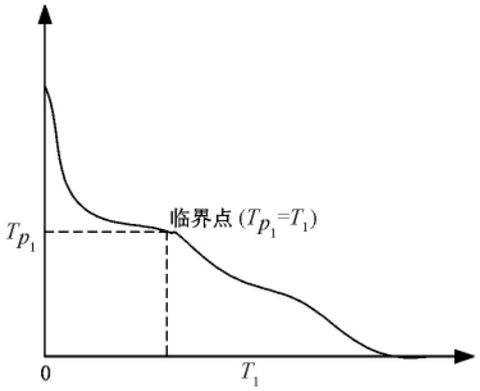

图 2　负载均衡调度时间曲线

如果充分利用计算节点的计算资源,提高任务并行能力作为最佳调度策略的基本依据,那么,Map/Reduce 的最佳调度发生在图 2 的临界点上,该点上满足 $T_{p_1} = T_1$.

据(4)式,所有节点可建立如下数组:

$$\left\{\begin{array}{l} T_1 = T_{p_1} \\ T_2 = T_{p_2} \\ \cdots \\ T_n = T_{p_n} \end{array}\right\}. \tag{5}$$

在一次最佳调度的 Split 任务中,$T_{p_j}(1 \leqslant j \leqslant n)$ 必然为某一个节点的运行时间,即

$$T_{p_j} \in \{T_i \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n\}. \tag{6}$$

据(5)式和(6)式,最佳调度策略满足:

$$T_1 = T_2 = \cdots = T_n. \tag{7}$$

(7)式可这样的解释: 一个时间响应最快的调度方式,需要所有节点充分的参与,基本没有间断.

为实现基于处理时间的动态均衡策略,需要为每一个计算节点 i 建立一个临时变量 $t_i(1 \leqslant i \leqslant n)$,初始为 0. 当计算节点提交 Map 任务之前,记录起始时间; 当计算节点完成 Reduce 任务后,将当前时间减去起始时间,得到子任务所用时间 $\triangle t$. $\triangle t$ 综合反映了目标计算节点的处理能力和负载能力. 令 $t_i = t_i + \triangle t$,t_i 为节点处理任务用去的总时间.

如果将新任务按节点用去时间最少分配,当无子任务返回时,t_i 均为 0,导致前期任务集中于第 1 个节点,从而降低任务执行效率. 在评价节点处理时间时,已经被分配但未完成的子任务应纳入处理总时间范畴. 假定未完成的任务按节点当前处理平均速度进行,设节点 i 当前已处理的任务数为 N_{h_i},已提交的任务数为 N_{s_i},节点 i 处理总时间 T_i 按(8)式计算:

$$T_i = \begin{cases} t_i / N_{h_i} \times N_{s_i}, & N_{h_i} > 0, \\ N_{s_i} \times t_0, & N_{h_i} = 0, t_0 > 0, \end{cases} \tag{8}$$

其中 t_0 为节点预估的单元任务执行时间,可预先设置为一个数值很小的正数,如 0.01.

据(8)式,任务调度策略为: 从节点序列中选择当前最小处理时间的节点. 当节点 i 未分配任务时,$t_i = 0$,因此任务调度之初按节点列表的先后顺序提交任务,之后按照最小处理总时间的策略调度. 由于 T_i 随着子任务的执行不断变化,该调度策略属于动态均衡调度策略.

4　算法评价

为证明算法的有效性,在局域网环境创建 2 个处理能力基本相同的计算节点. 在该环境下执行一个由 500 个计算单元组成的 Map 任务,分别设定各计算单元处理时间,如表 2 所示.

将本文算法与其它调度策略对比,包括基于平

均处理时间的动态均衡调度、基于已提交任务总数的动态均衡调度、发送者启动、接受者启动、单节点几种. 其中,前 2 种与本文算法类似,均为动态记录节点负载状态进行调度的方法,区别在于负载评价依据不同; 对于发送者启动策略,在分配任务时,首先从中心服务器中查询节点的最大响应时间; 在接受者启动策略中,节点在子任务完成后,提交节点的单元任务处理时间; 单节点是指各线程只提交给一个固定的节点处理.

表 2 不同调度策略的任务执行时间(s)

单元任务处理时间 /s	单节点	平均处理时间调度	提交任务数	发送者启动	接受者启动	本文调度算法
0.016	9.362	8.611	5.585	9.124	5.112	5.085 1
0.031	16.263	16.022	10.267	15.649	10.380	10.374 0
0.063	30.982	29.188	19.516	29.021	17.630	17.582 0
0.141	72.291	69.638	44.470	70.432	44.895	44.570 0
0.281	143.506	138.664	84.858	139.421	84.845	83.866 0
0.577	346.143	330.864	164.211	335.430	163.789	162.031 0

从表 2 可见,单节点由于只有一个节点参与整个计算,代表了调度极端情况,效率最低; 平均处理时间调度和发送者启动效率稍高; 提交任务数、接受者启动和本文调度算法效率最高; 由于在本文调度算法中,不存在节点与调度中心的负载信息传递,执行效率稍高于接受者启动.

为进一步研究各调度算法的特点,需要研究调度的具体细节. 设 2 个计算节点分别编码为 0 和 1; 由于 Split 子任务的分配顺序与定义顺序相同. 将 Split 按分配顺序记录各任务被分配的计算节点 D_i($i \leqslant 500$,$D_i = 0$ 或 1),那么 D_i 组成的序列反映了调度的具体细节. 在只有 2 个计算节点的条件下,时间分散度 div 按(9)式计算:

$$div = \sum_{i=1}^{n-1} | D_{i+1} - D_i | / n. \qquad (9)$$

据(9)式,div 取值范围在 0 ~ 1 之间,值越大,任务在 2 个节点间时间分配越均匀. 各调度算法的时间分散度如表 3 所示.

结合表 2 和表 3,可发现本测试中的任务执行时间与时间分散度成负相关关系. 由于平均处理时间调度不能正确反映节点当前负载状态、发送者启动策略不能及时反映节点负载,导致时间分散度值小,任务效率低下; 将基于提交任务数的调度与本文调度对比,发现 2 个算法效率相当,但前者机械地将任务数量作为负载指标,要求各节点执行的任务量大致相等. 由于忽略了各节点的实际处理能力,仅当各节点处理能力相当并且子任务复杂度相近时该调度策略才合理.

表 3 不同调度策略的时间分散度

单元任务处理时间 /s	单节点	平均处理时间调度	提交任务数	发动者启动	接受者启动	本文调度算法
0.016	0	0.01	1.00	0.01	0.75	0.74
0.031	0	0	1.00	0	0.78	0.79
0.063	0	0.03	1.00	0.02	0.80	0.80
0.141	0	0.04	1.00	0.03	0.80	0.81
0.281	0	0.04	1.00	0.03	0.81	0.81
0.577	0	0.04	1.00	0.03	0.81	0.81

5 结语

本文提出一种基于处理时间的 Map/Reduce 动态均衡调度调度算法. 该算法通过节点的处理总时间作为最佳候选计算节点的依据. 为避免任务初次调度时,子任务集中于第一个节点,将已分配的子任务按节点当前处理子任务的平均时间纳入节点处理时间. 由于节点的处理时间通过节点的实际任务处理过程进行动态评价,减少了节点与云中心服务器的负载信息传递,提高了 Map/Reduce 任务执行效率,其研究成果具有重要的理论意义和实用价值.

6 参考文献

[1] Yang Hung-chih, Ali Dasdan, Hsiao Ruey-lung, et al. Map-reduce-merge: simplified relational data processing on large clusters [EB/OL]. [2012-12-16]. http://www.cs.duke.edu/courses/cps399.28/current/papers/sigmod07-YangDasdanEtAl-map_map_reduce_merge.pdf.

[2] 王霜,修保新,肖卫东. Web 服务器集群的负载均衡算法研究 [J]. 计算机工程与应用,2004,40(25):78-80.

[3] Foto N, Afriti, Jeffrey D, et al. Optimizing joins in a map-reduce environment [EB/OL]. [2012-12-16]. http://infolab.stanford.edu/~ullman/pub/join-mr.pdf.

[4] 吴宝贵,丁振国. 基于 Map/Reduce 的分布式搜索引擎研究 [J]. 现代图书情报技术,2007,2(8):52-55.

[5] Berthold Jost, Dieterle Mischa, Loogen Rita. Implementing parallel Google map-reduce in Eden [J]. Euro-Par 2009 Parallel Processing, 2009, 5704: 990-1002.

[6] Jeffrey Dean, Sanjay Ghemawat. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[7] Kevin Barker, Andrey Chernikov, Nikos Chrisochoides, et al. A load balancing framework for adaptive and asynchronous applications [J]. Parallel and Distributed Systems, IEEE Transactions on, 2004, 15(2): 183-192.

[8] Arnaud Legand, He1e ne Renard, Yves Robert, et al. Mapping and load-balancing iterative computations [J]. Parallel and Distributed Systems, IEEE Transactions on, 2004, 15(6): 546-558.

[9] Helene Renard, Yves Robert, Frederic Vivien. Static load-balancing techniques for iterative computations on heterogeneous clusters [J]. Euro-Par 2003 Parallel Processing, 2003, 2790: 148-159.

[10] 蒋翠清,杨善林,黄梯云. 基于 Agent 的动态负载均衡技术及仿真实现 [J]. 微电子学与计算机,2005,22(10):47-50.

[11] 邓成玉 章,刘永山. 动态负载均衡策略及相关模型研究 [J]. 计算机工程与应用,2011,47(8):131-134.

[12] Willbebek-Lemair Marc H, Reeves Anthony P. Strategies for dynamic load balancing on highly parallel computers [J]. Parallel and Distributed Systems, IEEE Transactions on, 1993, 4(9): 979-993.

[13] Zhang Xiayu, Yu Yongquan, Chen Baixing, et al. An extension-based dynamic load balancing model of heterogeneous server cluster [EB/OL]. [2012-12-16]. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=243526&userType=inst.

[14] Taylor, Ronald C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics [EB/OL]. [2012-12-18]. http://www.biomedcentral.com/1471-2105/11/S12/S1.

[15] 刘晔,沈潇军,刘摩西. 基于云模式的负载均衡策略研究 [J]. 电脑与电信,2012,12:38-40.

An Algorithm for Map/Reduce Dynamic Loading Balancing Policy Based on Processing Time

CHEN Jun[1,2], LU Han-yu[1,3], YAO Dan-dan[1]

(1. Key Laboratory of Earth Exploration & Information Techniques of Ministry of Education, Chengdu University of Technology, Chengdu Sichuan 610059, China; 2. Institute of Resources and Environment, Chengdu University of Information Technology, Chengdu Sichuan 610225, China; 3. Department of Computer and Information Engineering, Guizhou University, Guiyang Guizhou 550003, China)

Abstract: An algorithm of dynamic loading balancing policy for Map/Reduce based on processing time has been proposed. In this algorithm, each computing node is assinged the process total time indicators. The processing time of each node is updated after it returns the result of subtask, which is the key index for selecting the best node. Experiments show that, due to the reduction of communication frequency between the node and the cloud server, the algorithm can take full advantage of the processing power of computing nodes and improve the Map / Reduce scheduling efficiency.

Key words: cloud computing; Map/Reduce; load balancing scheduling

(责任编辑: 冉小晓)