

DMS 算法在 Map/Reduce 任务调度中的应用

裴树军, 孔德凯, 苗 辉

(哈尔滨理工大学 计算机科学与技术学院, 黑龙江 哈尔滨 150080)

摘 要: 云环境下传统的任务调度算法整体效率较低, 为了提高任务调度的整体效率, 在 Map/Reduce 基础上提出了一种基于处理时间的 DMS 任务调度算法。首先, 对复杂任务进行预处理, 将复杂任务转化为 DAG 图, 依据任务依赖关系大小产生最佳拓扑排序, 并依据排序结果将复杂任务交给 work 节点进行处理; 其次, 通过将节点处理任务的预测时间与节点处理能力的比值作为子任务在每个节点的处理“时间”进行量化建模, 建立任务和处理时间的度量矩阵, 依据 DMS 算法进行处理, 从而获得任务分配最佳方案; 最后, 从任务调度效率与资源使用率的角度将 DMS 算法与公平调度算法、遗传算法行对比验证。实验结果表明, DMS 算法能明显提高任务调度整体效率, 充分利用各节点的计算能力提高了 Map/Reduce 的调度效率。

关键词: 云计算; Map/Reduce; 任务调度; 差值矩阵

DOI: 10. 15938/j. jhust. 2019. 01. 012

中图分类号: TP319 **文献标志码:** A **文章编号:** 1007-2683(2019)01-0071-07

DMS Algorithm in the Application of the Map/Reduce Tasks Schedule

PEI Shu-jun, KONG De-kai, MIAO Hui

(School of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China)

Abstract: The whole efficiency of traditional task scheduling algorithms is low under the cloud environment, In order to improve the whole efficiency of the task scheduling, this article based on Map/Reduce presents a Difference Matrix Scheduling tasks schedule algorithm based on processing time. Firstly, pretreatment of complex tasks, the complex tasks is converted to Directed Acyclic Graph figure, the tasks are topological sorted in an optimal manner according to the size of the task dependencies, and the work node is accordance with the sort to processing the complex tasks; Secondly, using the ratio of predictive time that node process tasks to node process capacity as a subtask in each node time quantitative modeling, then establish the task and the metric matrix of process time, according the Difference Matrix Scheduling to processing the matrix, and obtain the optimal scheme of task assignment. Finally, the experiment evaluates the Difference Matrix Scheduling, fair scheduling algorithm, genetic algorithm in the task scheduling and resource utilization efficiency angles. The results show that the algorithm can significantly improve the overall efficiency of complex task scheduling and make full use of the capacity of the compute nodes to improve the Map / Reduce scheduling efficiency.

Keywords: cloud computing; map/reduce; tasks assign; difference matrix

收稿日期: 2017-02-27

基金项目: 国家自然科学基金(60572153, 60972127)。

作者简介: 裴树军(1972—), 男, 博士, 硕士研究生导师;

苗 辉(1994—), 女, 硕士研究生。

通信作者: 孔德凯(1991—), 男, 硕士研究生, E-mail: kongdekai168@163.com.

0 引言

随着物联网、移动互联网、社会化网络的快速发展,数据来源的渠道逐渐增多,半结构化及非结构化数据呈几何倍增长,从而加速了大数据^[1-2]处理技术的快速发展与变革。云计算作为一种新兴的商业计算模式,采用并行的处理方式提高了大数据的处理效率。任务调度^[3-5]问题一直是云计算系统关注的核心问题,而影响任务调度效率的因素很多,其中任务调度模型与算法的好坏能够直接影响云计算系统的整体性能。现在很多学者都提出了很多有效的方法:国内的 Hadoop^[6]技术论坛的总编易剑等学者提出了 Map-Balance-Reduce 模型,即在 Map 节点处理完任务形成中间任务后,使用一个 balance 的循环过程进行均衡 Reduce 的输入,这样可以用来解决输入不平衡问题; Abhishek Verma 提出了一种 LATE 调度算法,该算法主要是通过计算待执行和正在执行任务的剩余时间,将执行最慢的任务进行备份从而缩短 Map/Reduce 作业执行时间; Tang Zhou 等提出了 MTSD 算法,该算法主要考虑数据的本地行与集群异构特点,并且以任务执行截止期限作为依据。依据节点计算能力大小决定数据存储的大小,提高了任务数据本地性。

在云平台上 Map/Reduce^[7]对数据分而治之,将大的任务分解成大量的子任务,再将子任务提交到 Task Tracker 来完成。由于每个计算节点的剩余空间和对不同任务处理效率不同,同样的任务如果没有一个合理的任务调度机制,在调度中没有合理的考虑复杂任务的处理过程,可能会因为某一个小的任务的处理时间过长而导致整个任务处理效率较低,因此对任务的合理调度是提高任务处理效率的关键。本文主要针对多任务调度提出了一种在 Map/Reduce 框架下基于差值矩阵的 DMS (Difference Matrix Scheduling) 的任务调度算法,以抽样法获得节点处理任务的时间,再将其与节点处理任务的能力^[8-9](计算能力、网络带宽能力等)的比值作为时间预测,即处理时间与节点处理能力成反比,而抽样的任务处理时间与真实任务处理时间成正比,其中节点处理能力 $\Gamma_{ij} = t_{i,l}/r_{j,c}$ 为任务长度与资源计算能力的比值。将得到的任务处理时间进行量化,建立任务和节点处理时间的 $n \times n$ 矩阵,使用 DMS 算法一步步缩小任务规模获得局部最优解,最终获得任务分配的全局最优解。对于复杂任务首先对其

进行复杂任务的初始化,将复杂任务转化为 DAG 图,在进行最佳拓扑排序,然后依次分配给 work 节点进行处理,最后将得到的最终结果与简单任务一起使用 DMS 算法进行处理。其中该 DAG 图与传统算法相比每个节点都充分考虑内存资源的使用情况,从而减少了因不确定因素导致的内存溢出产生的任务执行失败的情况;并且任务之间的依赖关系是任务排序的根据,这样很大程度上使任务之间的关联更明确;如果某个任务执行失败,可以根据依赖关系的大小依次从父节点的内存中取出需要的信息与现有内存数据一起根据节点负载重新执行该任务,因此提升了任务的调度效率和容错性能。通过使用 Hadoop 平台进行任务调度实验,证明该算法能够有效地提高任务调度的整体效率,具有一定的理论和实际应用价值。

1 Map/Reduce 任务调度过程及问题分析

Map/Reduce 任务调度过程^[10-12]可以简单的抽象为 Map(映射)和 Reduce(规约)两个基本操作,如图 1 所示。在 Map 阶段,每个 Map 节点读取相应的子任务(split),使用 Map 函数对子任务进行处理解析出中间 Key/Value 对,并将中间结果进行 Merge 后存储在本地磁盘;在 Reduce 阶段,Reduce worker 根据位置读取 Map 产生的中间结果,调用 Reduce 函数对中间结果进行处理,并将结果添加到最终输出文件中。通过对任务调度过程分析可知,影响整个任务调度算法效率的因素主要包括两部分,第一是任务调度过程中复杂任务的处理效率;第二个是整个过程中 I/O 操作的密集程度,即内存与磁盘以及磁盘与磁盘之间的数据交换密集程度(Shuffle 过程最为密集)。

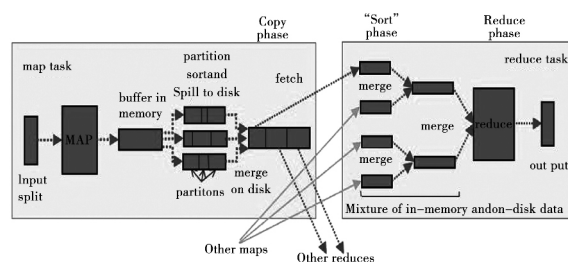


图1 Map/Reduce 模型

Fig. 1 Map/Reduce mode

因此本文基于原有的 Map/Reduce 模型通过在 Map 节点之前增加 work 节点,如图 2 所示,对模型

进行改进。通过将输入的任务按照简单任务和复杂任务进行分类,把复杂任务交给 work 节点进行预处理。由 Master 节点将复杂任务表示为 DAG 图,根据任务之间的依赖关系产生最佳拓扑排序,之后将排好序的子任务集合依次分配给中间的 work 节点,其中 work 节点类似于 Map 节点按照顺序的执行复杂任务,形成简单任务传递给 Map 节点。任务调度过程中在任务调度 Map 端创建内存资源池,收集 Map 处理中因为 Spill 溢写而产生的数据,在 Map 任务执行结束后将所有结果一起汇总写入中间结果集,在 Reduce 端增加 Shuffle 内存缓存区大小并提高阈值,使 Copy 来的数据直接在内存中进行 Merge,直到数据 Merge 结束传入 Reduce 输出最终结果。通过改进使数据在内存中交换处理,从而减少了内存与磁盘以及磁盘与磁盘之间的数据交换。整个 Map/Reduce 任务调度过程不用考虑复杂任务的相互制约以及内存磁盘数据交换时间过长问题,在任务执行失败时,可以直接根据最佳拓扑排序回溯到依赖关系最强的父节点获取所需数据,减少遍历所有依赖关系节点所需的时间,从而提高任务调度的整体效率。

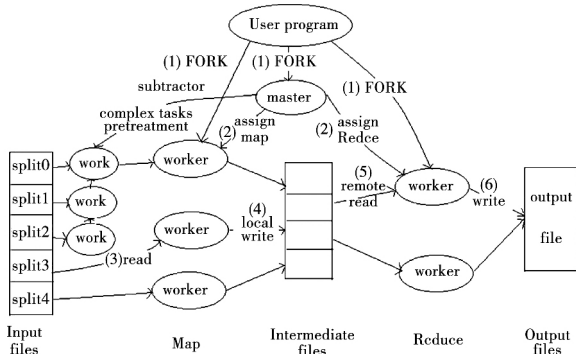


图 2 改进 Map/Reduce 模型

Fig. 2 Improved Map/Reduce mode

2 任务调度理论和步骤

2.1 DMS 任务调度算法理论

云环境下 DMS 任务调度算法^[3],依据 Map 节点处理子任务的预测时间与节点计算能力的比值进行量化,建立 Map 节点与子任务之间的 $n \times n$ 时间矩阵,依据该矩阵进行任务分配。DMS 任务调度算法的实质是实现任务与 Map 节点之间的最佳分配方案,达到任务调度的效率最优以及节点资源合理利用且执行时间最短的目的。因此任务调度过程必须满足两个约束条件:第一所有任务都要执行,第二

每个 Map 节点同时只能执行单个子任务。

设矩阵 $A = (a_{ij})_{n \times n}$ ($a_{ij} = 0$ 或 1) 为任务与节点之间的分配矩阵(初始值为 0),元素 a_{ij} 的值为 1 或 0,1 为 true,0 为 false。记矩阵 $V = (x_{ij})_{n \times n}$ 为任务分配的时间代价矩阵, x_{ij} 表示第 j 个节点处理第 i 个任务所用的时间代价值。所以最优的任务调度分配可以表示为

$$F_{\min}^{a \in T}(A, V) = \sum_{i=1}^n \sum_{j=1}^n (a_{ij} \times x_{ij}) \quad (1)$$

其约束条件为

$$\sum_{i=1}^n a_{ij} = 1 \quad (j = 1, 2, 3, \dots, n) \quad (2)$$

即一个 Map 节点完成一个任务

$$\sum_{j=1}^n a_{ij} = 1 \quad (i = 1, 2, 3, \dots, n) \quad (3)$$

即一个任务只能被一个 Map 节点完成。

我们记矩阵

$$FS = (f_{ij})_{n \times \frac{n(n-1)}{2}} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & \cdots & f_{1\frac{n(n-1)}{2}} \\ f_{21} & f_{22} & f_{23} & \cdots & f_{2\frac{n(n-1)}{2}} \\ \vdots & \vdots & \vdots & & \vdots \\ f_{n1} & f_{n2} & f_{n3} & \cdots & f_{n\frac{n(n-1)}{2}} \end{bmatrix}$$

为先序差值矩阵,即矩阵 V 中互不相同的两列的差值乘以前一列值所形成的矩阵。记矩阵

$$LS = (l_{ij})_{n \times \frac{n(n-1)}{2}} = \begin{bmatrix} l_{11} & l_{12} & l_{13} & \cdots & l_{1\frac{n(n-1)}{2}} \\ l_{21} & l_{22} & l_{23} & \cdots & l_{2\frac{n(n-1)}{2}} \\ \vdots & \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n\frac{n(n-1)}{2}} \end{bmatrix}$$

为后序差值矩阵,即矩阵 V 中互不相同的两列的差值乘以最后一列值所形成的矩阵。

2.2 相关定义

定义 1 $F = F_{\min}^{a \in T}(A, V)$ 表示任务处理时间最短

且效率最优分配函数,其中 T 为所有可能的任务分配方案集合,任务分配矩阵 $A \in T$,其中

$$a_{ij} = \begin{cases} 1 & R_i \rightarrow \text{map}_j \\ 0 & \text{others} \end{cases}, j = 1, 2, \dots, n \quad (4)$$

$R_i \rightarrow \text{map}_j$ 表示第 i 个任务被第 j 个节点处理。

定义 2 差值矩阵: FS 先序差值矩阵,即对 V 中所有不相同的两列做差,将差值乘以前一列的数

值 $s_{ik} = (x_{it} - x_{im}) x_{ik} \quad k = 0, 1, \dots, \frac{n(n-1)}{2} \quad i = 1, 2, \dots,$

$nt \neq mt, m = 1, 2, \dots, n$,其中 k 的值依次递增。 LS 后

序差值矩阵,即对 V 中所有不相同的两列做差,将差值乘以最后一列的数值 $l_{ik} = (x_{it} - x_{im}) x_{im} \quad k = 0, 1,$

$\dots, \frac{n(n-1)}{2} - i = 1, 2, \dots, nt \neq mt, m = 1, 2, \dots, n$, 其中 k 的值依次递增, 两个差值矩阵是进行任务筛选的必要条件。

定义3

$$F_{FS} = \min_{A \in T_1} F_{\min}(A_1, V) \quad (5)$$

$$F_{LS} = \min_{A \in T_1} F_{\min}(A_2, V) \quad (6)$$

分别为先序差值矩阵和后序差值矩阵分配最优解局部最优目标函数值。 A_1, A_2 计算如下:

$$A_1 \parallel A_2 = A_{ij} = \begin{cases} 1 & i=r \text{ 且 } j=t \\ 0 & \text{其它} \end{cases} \quad (7)$$

对于 A_1 , r 为 FS 先序差值矩阵中最小值对应的行号 r , t 为对应前一列的列号 k :

$$r = \min_i (x_{ik} - x_{im}) x_{ik} \quad (8)$$

对于 A_2 , r 为 LS 后序差值矩阵中最小值对应的行号 r , t 为对应的前一列好的列号 k :

$$r = \min_i (x_{ik} - x_{im}) x_{im} \quad (9)$$

其中两式均满足: $i = 1, 2, \dots, n \quad k \neq m, k, m = 1, 2, \dots, n$, 这样形成了两个暂时最优分配矩阵 A_1, A_2 。并且确保形成的局部任务分配矩阵函数值小于等于全局分配矩阵函数值:

$$F_{\min}(A_1, V) \leq F_{\min}(A', V), F_{\min}(A_2, V) \leq F_{\min}(A', V)$$

以后 A_1, A_2 中的 0 元素将不再进行任务分配。设定以

$$a_{rt} = 1, \begin{cases} a_{rj} = 0 & j = 1, 2, \dots, n, j \neq t \\ a_{it} = 0 & i = 1, 2, \dots, n, i \neq r \end{cases} \quad (10)$$

对 A 进行任务剔除。这样就很明显可以看出 V 变为一个 $(n-1)$ 阶的任务分配问题, 则任务的分配空间将缩小, 依次进行任务的剔除直到矩阵 V 只剩下一个元素为止, 最后一个任务将直接进行分配。

定义4 全局最优分配矩阵 $A' = \min(\max(A_1), \max(A_2))$, 其中 $\max(A_1), \max(A_2)$ 表示矩阵 A_1 和 A_2 中元素的最大值, 即为任务并行处理所用的总时间, 然后取两个时间中最小的所对应的矩阵就是最佳任务分配矩阵, 按照该分配方式对任务进行分配即可得到最优分配。

定义5 复杂任务 $(T, \rightarrow, A)$: 居于依赖关系的任务组合, 即在子任务之间居于一定的偏序关系。其中 $T = (t_1, t_2, \dots, t_n)$ 是一组具有依赖关系的子任务, $\rightarrow$ 用来表示子任务之间所具有的优先约束关系, 如: $t_i \rightarrow t_j$, 则表示 t_i 必须在 t_j 之前开始执行, M 表示子任务的计算量, $M_i > 0 (1 \leq i \leq n)$ 为第 i 个任务的计算量。

2.3 复杂任务的预处理

云计算中, 组成复杂任务的子任务之间存在一定的优先约束关系, 所以可以将复杂任务表示为一个 DAG (directed acyclic graph) 图, $V = (T, E)$, 其中 T 表示图中节点的集合, E 表示边的集合, 边的权值表示节点之间的依赖关系, 权值越大依赖关系越大, 如果权值为 0 则两个任务可并行处理。DAG 图的优先约束关系可以表示为一个节点在没有获得其前序节点信息之前是不能执行的。首先由 Master 节点根据任务之间的关联关系以及关联的程度, 将复杂任务转换为 DAG 图, 如图 3 所示。再将 DAG 图用拓扑排序转换为最佳拓扑排序 (OTS, optimum topological-sort), 如图 4 所示。其中 DAG 图中 V 的节点数与 T 中节点数相同, V 中有足够的边能连接所有节点且没有回路。由 Master 节点按照 OTS 顺序, 对每个子任务的优先顺序进行排序, 实现复杂任务执行顺序的初始化。

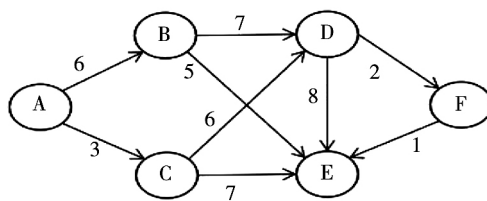


图3 DAG图

Fig.3 DAG graph

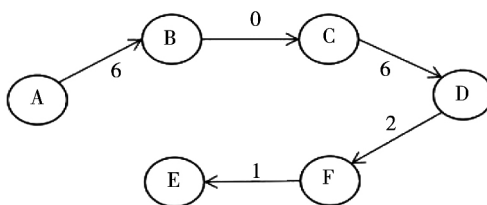


图4 OTS

Fig.4 OTS

然后由 Master 节点根据 work 节点的计算能力将初始化后的任务分配给 work 节点。此时的 work 节点具有两个特点: ①子任务存在偏序关系, work 节点的执行不具有并行性, 需要按流程依次执行。②任务的执行是由 Master 在执行 Map 节点任务之前分配的。最后当所有复杂任务的子任务执行完后, 以简单任务的形式与没有依赖关系的子任务一起由 Master 分配执行任务调度过程。

2.4 DMS 任务调度算法具体步骤

首先由 Map/Reduce 库函数对用户提交的任务请求进行划分, 对生成的子任务进行识别、分类, 将具有依赖关系的子任务分为一类, 简单任务分为一类; 其次将具有依赖关系的子任务交由 work 节点进

行复杂任务预处理,形成复杂任务结果,将形成的结果与简单任务一起按照 DMS 算法进行分配,具体过程如下:

1) 由 Master 节点将任务分配矩阵 A 进行初始化,令 $a_{ij}=0 \quad i,j=1,2,\dots,n$,即所有任务分配状态为 false 不可执行状态。

2) 由 Master 调用 work 减法器按照定义 3 分别求出对应的 FS 和 LS 即: $FS = s_{ik} = (x_{it} - x_{im}) x_{it} LS = l_{ik} = (x_{it} - x_{im}) x_{im} \quad k=0,1,\dots,\frac{n(n-1)}{2} \quad i=1,2,\dots,nt \neq mt, m=1,2,\dots,n$ 。

3) 对 FS 进行任务选取: 由 Master 节点对 FS 进行遍历选取,选取出矩阵中最小值(如果存在相同的则取第 1 个),提取出对应的 (r,t) 即: $r = \min_i (x_{it} - x_{im}) x_{it} \quad i=1,2,\dots,n \quad t \neq mt, m=1,2,\dots,n$ 的行号 r ,和第 1 个元素的列号 t ,令 A_1 中 $\alpha_{rt}=1$ 即将该任务状态设置为 true。

4) 形成临时最优分配矩阵 A_1 : 将选取的坐标 (r,t) 对应初始化矩阵 A ,将对应元素值改为 1 即为 true,该元素对应的行和列值不变, $\begin{cases} a_{ij}=0 & j=1,2,\dots,n, j \neq t \\ a_{it}=0 & i=1,2,\dots,n, i \neq r \end{cases}$,则该坐标对应的行和列除本元素外全部删除掉,使矩阵规模减小一阶^[14-15]。同时将 FS 中 (r,t) 对应的行和列进行剔除,对剩余矩阵中元素进行提取(如果得到的 r 与已分配的相同则该值不符合,依次获取剩下的最小值直到新取的 r 与已分配的不同)找到对应的 (r,t) ,重复执行 3) 4) 直到矩阵只剩一个元素,这样只需进行 $(n-1)$ 次操作就可以获得 A_1 。

5) 根据 A_1 对时间代价矩阵 V 进行判断: 检查 A_1 所对应 V 中的值,在当前列为最小值的列的个数,如果个数大于 $n/2$ 则 A_1 为最优分配矩阵 A ,转入步骤 8),否则转入步骤 6)。

6) 对 LS 进行任务选取: 按照步骤 3) 进行选取将提取公式改为: $r = \min_i (x_{it} - x_{im}) x_{im}$ 其余保存不变,在按照步骤 4) 生成临时最优分配矩阵 A_2 。

7) 获得最优分配矩阵 A : 根据时间代价矩阵 V 将分配矩阵 A_1 和 A_2 所对应的任务处理总时间进行对比,处理时间短的分配矩阵为最佳任务分配矩阵 A :

$$A = \min(\max(A_1), \max(A_2))。$$

8) 到此所有的任务都进行了最优分配,对每个 Map 节点对应的任务进行合理的分配,最后按照最

优分配方案 A 由 Master 交给对应的 Map 节点进行处理。

9) 这样简单任务、复杂任务经 DMS 算法处理后产生了最优的任务分配方案,并按此方案进行最终任务调度。在 Map 端首先为每个处理机根据 Map 节点的数量与内存缓冲区大小建立适当的内存资源池(大于单个 Map 节点内存缓冲区大小,且可适当调整),然后将任务传入 Map 节点进行任务处理,当产生的数据量达到 Map 节点内存缓存区阈值时,将 Map 节点产生的键值对进行 Merge 并传入内存资源池,直到所有 Map 任务处理结束开启进程将数据写入本地磁盘进行备份。

10) 在任务调度过程中由 TaskTracker 实时监测每个任务的执行状态并进行实时纠错,如果任务执行成功则将每个 Map 节点数据缓存区数据存入资源池,并记录在资源池的存储位置,再将其恢复到初始状态并请求下次执行任务;如果任务处理失败,对于复杂任务保留当前节点数据并根据 Master 节点存储的每个任务的 OTS 以及任务依赖关系的大小,依次回溯到当前 work 节点的父节点获取父节点在资源池中数据,直到在父节点中获取的数据能够满足当前任务的数据需求为止,结合当前任务数据并按照节点处理能力重新分配任务节点进行任务处理,对于简单任务直接根据节点处理能力重新对任务进行分配处理,直到任务执行成功。

11) 在 Map 节点任务执行完成后,向 JobTracker 发送任务完成指令,然后执行 Reduce 前的 Shuffle 过程。此时 Shuffle 的缓存区容量远远大于 Reduce 节点的缓存区,将所有已完成的 Map 节点缓存区的数据直接拉取 Copy 到 Shuffle 缓存区,并将获取的数据进行 Merge,直到获取所有的已完成的 Map 节点数据为止,此时调用进程将产生数据存入磁盘进行备份,同时将数据传入 Reduce 端进行最后的 Reduce 过程并将最终产的文件存入 HDFS,整个任务调度过程完成。

由 DMS 算法具体步骤可知,对于解决 N 个任务的分配问题,只需要进行 $(n-1)$ 次迭代循环即可得出最优解。该过程将复杂任务与简单任务分开处理,很大程度上减少了因为复杂任务之间依赖关系而导致的作业执行失败的情况。并且该过程减少了大量的数据 I/O 操作,使数据能够在内存中进行交换,提升程序的运行效率。根据对 Map/Reduce 模型的分析并结合任务的处理过程实现了对原有的任务调度的合理优化,提高了任务调度的整体效率。

3 算法的验证与比较

为了检验预测时间的准确性以及算法的效率,该部分采用 Hadoop 平台的基准测试^[6]进行实验验证,将该算法与公平份额调度算法^[7]以及遗传算法^[8-19]进行对比验证。

实验环境: 本实验使用 Hadoop1.2.1,由6个节点组成的集群环境,其中一个为 Master 节点,其余5个为 slaver 节点,机群之间是同构的,具体的集群配置情况见表1,在试验中为集群配置了默认的负载均衡^[2]策略,并且采用 Ganglia 来监测集群状态,主要是监测内存利用率。

表1 集群相关配置情况

Tab.1 Cluster related configuration

项目	配置
Hadoop 版本	Hadoop1.2.1
操作系统	CentOS6.5 64 位
JDK	1.7
Master	双核 4GB 500GB 硬盘 1
Slaver	双核 2GB 500GB 硬盘 5

实验 1: 使用 Hadoop 自带的基准测试程序 wordcount 和 terasort,证明使用抽样法对 Map 节点的处理任务进行时间预测是可行的。为了确保结果的正确性,最终数值采用 4 次运行程序的平均值作为最终值。在不同数目的 Map 下,独立运行 wordcount 和 terasort 程序进行对比,如图 5 所示。由图可以得出随着 Map 节点数目的增加,任务的执行时间一直维持在一个稳定的范围内变化较小,这就说明使用少量 Map 节点进行任务处理时间的预测是合理的。并行任务处理任务之间并没有明显的依赖关系,所以子任务之间的关系对 Map 节点时间预测影响不大,所以可以进行合理预测。

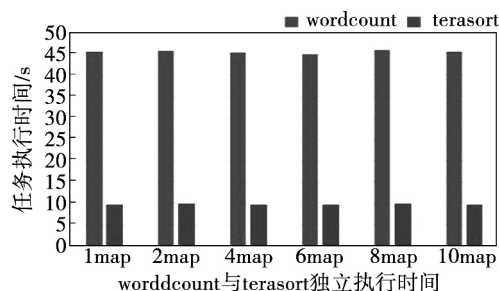


图5 任务独立执行时间

Fig.5 Task independent execution time

实验 2: 对比实验分别从任务调度的整体效率和资源使用率角度来证明 DMS 算法的优越性。其

中任务调度效率通过任务调度的平均响应时间和任务的整体执行时间以及不同算法下的吞吐量进行实验;使用内存使用率衡量系统的资源使用率。分别采用 100,150,200,⋯,350 个任务进行任务调度处理,3 种不同算法的平均响应时间以及任务执行时间对比如图 6、7。由图 6 可以得出 3 种算法的任务调度平均响应时间都随着任务数量的增加在逐步增加,在任务较少时 3 种算法之间的差距较小,但随着任务数量的增加 DMS 算法的平均响应时间明显低于公平调度算法和遗传算法。由图 7 可以看出,任务调度的执行时间随着任务数量的增加,DMS 算法的执行时间明显小于公平调度算法和遗传算法。

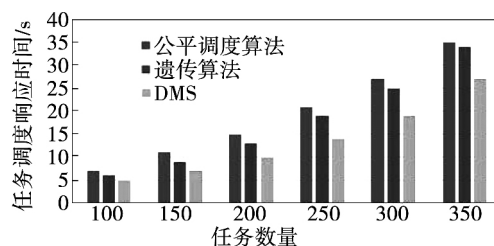


图6 任务调度平均响应时间

Fig.6 Task scheduling average response time

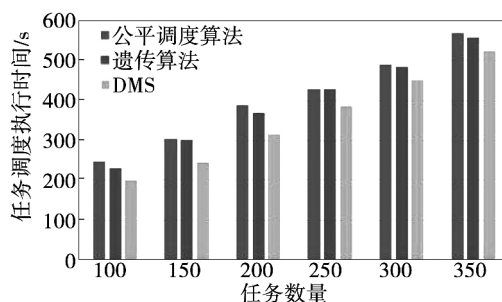


图7 任务调度执行时间

Fig.7 Task scheduling execution time

公平调度算法、遗传算法以及 DMS 算法进行任务调度处理后得到的平均吞吐量对比如图 8 所示。

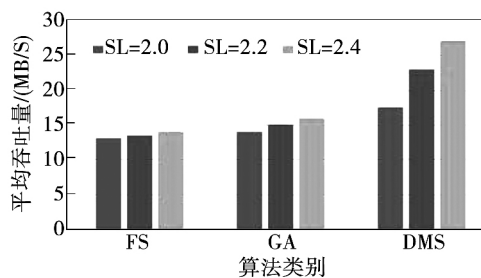


图8 不同算法的平均吞吐量

Fig.8 The average throughput of different algorithms

由图 8 可知,DMS 算法的吞吐量在不同的数据流负载量下,均要高于其它的 3 个算法,其中公平调度算法以及遗传算法的实验结果随着数据流负载量

的改变,没有太大波动。当流量负载参数 SL (Stream Load) 从 2.0 增加到 2.4 时,DMS 算法的平均吞吐量有很大的提升,说明 DMS 算法的任务请求与吞吐量之间是一种正相关,这也证明了 DMS 算法具有较强的任务处理能力。

在资源利用率方面,分别选取 3 种算法在任务调度过程中 100 ~ 500 s 时间内存资源使用情况进行对比,如图 9 所示,而开始和结束时段的内存使用率变化较大所以没有进行选取。从图中折线的变化可以看出,由于任务的大小以及内存使用情况等因素的影响,导致随着时间的增加内存使用率一直处在波动状态。因为 DMS 算法在任务调度过程中,将内存与磁盘以及磁盘与磁盘之间的数据交换过程转化为内存之间的数据交换,所以增加了内存的使用,所以整个任务调度过程中 DMS 算法的内存使用率要高于其它两个算法。综上所述,从任务调度的平均响应时间、任务调度的执行时间以及资源的使用率 3 个角度分析,得出 DMS 算法增加了任务调度的效率,提高了资源的利用率。

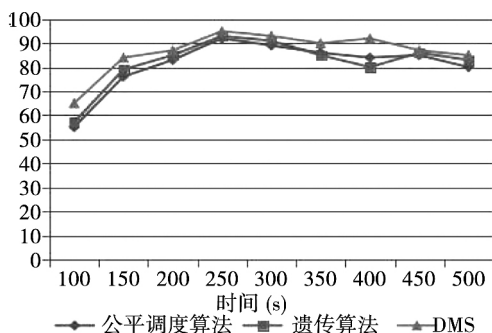


图9 内存利用率

Fig.9 Memory utilization

4 结 论

本文主要论述了在 Map/Reduce 框架下的任务的处理过程,通过 Master 对用户输入的作业进行分割,并按照任务之间的依赖关系进行任务分类。对于复杂任务,由 Master 节点转化为 DAG 图,然后形成最佳拓扑排序后再由扩展节点进行预处理。对于简单任务则采用 DMS 算法进行处理,建立任务与节点之间的时间矩阵,按照先局部最优到整体最优的方式进行任务分配。此外在任务调度过程中,采用增加内存资源池与增大 Shuffle 缓存区大小等方法,使数据在内存中处理,减少了因内存与磁盘以及磁盘与磁盘之间的数据交换而导致的处理时间过长、任务调度效率低的问题,极大地提高任务调度效率。

使用 Hadoop 平台分别从任务调度效率与资源使用率角度,将 DMS 算法与公平调度算法和遗传算法进行对比,实验结果表明 DMS 算法能明显提高任务调度整体效率,具有一定的理论指导意义。

参 考 文 献:

- [1] 黎建辉,沈志宏,孟小峰. 科学大数据管理: 概念、技术与系统 [J]. 计算机研究与发展, 2016, 54(2): 235.
- [2] 杨刚,杨凯. 大数据关键处理技术综述 [J]. 计算机与数字工程, 2016, 44(4): 694.
- [3] 史恒亮. 云计算任务调度研究 [D]. 南京: 南京理工大学, 2015.
- [4] 杨志伟,郑焱,王嵩,等. 异构 Spark 集群下自适应任务调度策略 [J]. 计算机工程, 2016, 42(1): 31.
- [5] ZAREI Mohammad Hossein, SHIRSAVAR Milad Azizpour, YAZDANI Nasser. A QoS-Aware Task Allocation Model for Mobile Cloud Computing [C] // 2nd International Conference on Web Research, Tehran, Iran, April 28, 2016, Institute of Electrical and Electronics Engineers Inc, 2016: 43.
- [6] 李德有,赵立波,解晨光. Hadoop 构建的银行海量数据存储系统研究 [J]. 哈尔滨理工大学学报, 2015, 20(4): 60.
- [7] 杜江,张铮,张杰鑫,等. MapReduce 并行编程模型研究综述 [J]. 计算机科学, 2015, 42(6A): 537.
- [8] 谢丽霞,严焱心. 云计算环境下的服务调度和资源调度研究 [J]. 计算机应用研究, 2015, 32(2): 528.
- [9] 宋杰,刘雪冰,朱志良. 一种能效优化的 MapReduce 资源比模型 [J]. 计算机学报, 2015, 38(1): 59.
- [10] LIN J C, LEU F Y, CHEN Y P. Impact of Map Reduce Policies on Job Completion Reliability and Job Energy Consumption [J]. IEEE Transactions on Parallel & Distributed Systems, 2015, 26(5): 1364.
- [11] 张红,王晓明,曹洁,等. Hadoop 云平台 MapReduce 模型优化研究 [J]. 计算机工程与应用, 2016, 52(22): 22.
- [12] 徐焕良,翟璐,薛卫,等. Hadoop 平台中 MapReduce 调度算法研究 [J]. 计算机应用与软件, 2015, 32(5): 1.
- [13] ARR Neto, ARR Neto. A New Pruning Method for Extreme Learning Machines Via Genetic Algorithms [J]. Elsevier Science Publishers B. V., 2016, 44: 101.
- [14] 马月坤,刘鹏飞,张振友,等. 改进的 FP-Growth 算法及其分布式并行实现 [J]. 哈尔滨理工大学学报, 2016, 21(2): 20.
- [15] 郑伟,马楠. 一种改进的决策树后剪枝算法 [J]. 计算机与数字工程, 2015, 43(4): 960.
- [16] 张新玲,颜秉珩. Hadoop 平台基准性能测试研究 [J]. 软件导刊, 2015, 14(1): 30.
- [17] 冯兴杰,贺阳. 改进的 Hadoop 作业调度算法 [J]. 计算机工程与应用, 2016, 53(12): 85.
- [18] 王波,张晓磊. 基于粒子群遗传算法的云计算任务调度研究 [J]. 计算机工程与应用, 2015, 51(6): 84.
- [19] 胡艳华,唐新来. 基于改进遗传算法的云计算任务调度算法 [J]. 计算机技术与发展, 2016, 10(16): 137.
- [20] 万聪,王翠荣,王聪. MapReduce 模型中 reduce 阶段负载均衡分区算法研究 [J]. 小型微型计算机系统, 2015, 36(2): 240.

(编辑: 温泽宇)