软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(12):3385−3398 [doi: 10.13328/j.cnki.jos.005236]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

关联性驱动的大数据处理任务调度方案*

王 玢, 吴雅婧, 阳小龙, 孙奇福

(北京科技大学 计算机与通信工程学院,北京 100083)

通讯作者: 阳小龙, E-mail: yangxl@ustb.edu.cn

摘 要: 目前大数据处理过程较少关注任务所处理数据间的依赖关系,在任务执行过程中可能产生大量数据迁移,影响数据处理效率.为减少数据迁移,提升任务执行性能,从数据关联性及数据本地性两个角度出发,提出了一种数据关联性驱动的大数据处理任务优化调度方案:D^3S^2(data-dependency-driven scheduling scheme).D^3S^2 由两部分组成:(1) 数据关联性感知的数据优化放置机制(dependency-aware placement mechanism,简称 DAPM),根据日志信息挖掘数据关联性,进而将强关联的数据聚合并放置于相同机架上,减少了跨机架的数据迁移;(2) 数据迁移代价感知的任务优化调度机制(transfer-aware scheduling mechanism,简称 TASM),完成数据放置后,以数据本地性为约束,对任务进行统一调度,最小化任务执行过程中的数据迁移代价.DAPM 和 TASM 互相提供决策依据,以任务执行代价最小化为目标不断迭代调整调度方案,直至最优任务调度方案.在 Hadoop 平台上进行的实验结果表明:较之原生 Hadoop,在不增加作业完成时间的基础上,D^3S^2 减少了作业执行过程中的数据迁移量.

关键词: 数据关联性;数据本地性;数据放置;任务调度;迁移代价感知

中图法分类号: TP316

中文引用格式: 王玢,吴雅婧,阳小龙,孙奇福.关联性驱动的大数据处理任务调度方案.软件学报,2017,28(12):3385−3398. http://www.jos.org.cn/1000-9825/5236.htm

英文引用格式: Wang B, Wu YJ, Yang XL, Sun QF. Dependency-Driven task scheduling scheme of big data processing. Ruan Jian Xue Bao/Journal of Software, 2017,28(12):3385−3398 (in Chinese). http://www.jos.org.cn/1000-9825/5236.htm

Dependency-Driven Task Scheduling Scheme of Big Data Processing

WANG Bin, WU Ya-Jing, YANG Xiao-Long, SUN Qi-Fu

(School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

Abstract: Currently, there is lack of consideration of dependencies between data in big data processing, resulting in low data processing efficiency with large amounts of data transfer during task execution. In order to reduce data transfer and improve processing performance, this paper proposes a data-dependency driven task scheduling scheme, named D^3S^2, for big data processing. D^3S^2 is mainly composed of two parts: dependency-aware placement mechanism (DAPM), and transfer-aware task scheduling mechanism (TASM). DAPM discovers dependency between data so that strongly related data will be clustered and assigned to nodes in the same rack, thereby reducing the cross-rack data migration. TASM schedules tasks simultaneously after data placement according to the data locality constraint, so as to minimize the data transfer cost during the task execution. DAPM and TASM provide basis for decision making to each other, iterating constantly to adjust the scheduling scheme with the goal of minimizing the execution cost until an optimal solution is reached. The proposed scheme is verified in Hadoop environment. Experiments show that compared to native Hadoop, D^3S^2 reduces the data transfer during job execution, and shortens job running time.

* 基金项目: 国家高技术研究发展计划(863)(2013AA01A209); 国家自然科学基金(61172048, 61303250)

Foundation item: National High Technology Research and Development Program of China(863) (2013AA01A209); National Natural Science Foundation of China (61172048, 61303250)

收稿时间: 2016-07-04; 修改时间: 2016-09-02, 2016-12-07; 采用时间: 2016-12-14; jos 在线出版时间: 2017-03-24

CNKI 网络优先出版: 2017-03-24 15:31:40, http://kns.cnki.net/kcms/detail/11.2560.TP.20170324.1531.005.html

Key words: data-denpendency; data-locality; data placement; task scheduling; transfer-cost-aware

近年来,随着数据量的不断攀升,MapReduce 等大数据处理框架在处理数据密集型应用方面发挥着越来越大的作用.不可忽视的是:在实际应用中,部分业务的输入数据之间存在关联性,这种关联性既可以表现为几个数据块可能被同一任务处理,也可表现为下一阶段的输入数据包含上一阶段的中间结果.以社交网络中用户画像信息为例,处于同一社交圈或在某方面有共同属性的用户,其相互之间的数据信息即存在关联性[1].在进行用户数据分析时,若对此类用户信息进行合并/聚合处理,则可有效减少任务输出结果大小,进而减少任务执行过程中的数据迁移.

现有的大数据处理框架在处理关联性数据方面并未考虑数据中心带宽资源的有限性,其采取的数据放置策略虽然保证了全局数据分布的均衡,但相关性较强的数据被随机放置于数据中心的节点上,可能在节点拉取数据阶段产生大量不必要的数据迁移[2].此外,现有任务调度机制仅考虑向 Master 请求任务调度的 Slave 是否满足完成任务所需资源约束,并没有对任务执行阶段(如 Shuffle 阶段)的数据迁移量做出有效预判,可能导致 IO 压力过大[3].

因此,如何减少不必要的数据迁移,是提升大数据框架处理性能的关键.为解决上述问题,本文从输入数据与处理任务之间的关联性出发,寻找输入数据间的关联性;进而通过优化数据放置策略,完成输入数据到数据中心节点的映射,最大程度实现数据本地化;同时,在数据本地性的约束下,以数据迁移代价最小化为目标,进行任务统一调度.

本文第 1 节给出相关工作的介绍,进一步阐明本文与相关工作的差异及研究意义.第 2 节给出大数据处理的任务调度问题描述及一般性建模.第 3 节进一步对数据关联性驱动的任务调度问题进行建模,并分析其具体机制:(1) 数据关联性感知的数据优化放置机制;(2) 数据迁移代价感知的任务优化调度机制.第 4 节通过实验对优化方案的性能进行分析.最后,对全文进行总结.

1 相关工作

对于 MapReduce 等大数据处理框架最为典型的数据密集型应用来说,大量的数据迁移会显著影响计算性能.因此,以数据本地性为中心改进调度策略,成为优化任务调度方案的重要手段之一.

为了提升任务执行性能,当前已有部分研究成果关注数据密集型应用中的数据关联性问题,即:从数据本身特点(如访问频率、位置等因素)及数据间相关性出发,将相关度高的数据聚合放置,以减少数据传输代价[4].针对此类问题,一些研究者提出了共享数据的概念.对于共享数据,Gu 等人提出对访问频率超过相应阈值的数据块创建新副本[5];而 Abad 等人提出将经常访问的数据直接放置在本地缓存中[6].虽然上述针对共享数据的处理机制在一定程度上减小了不必要的网络带宽消耗,但新增副本、创建缓存窗口等行为带来了创建副本的通信开销及本地缓存的存储开销,影响了节点的性能.在此基础上,另外一些研究者从数据关联性本身出发,给出了感知任务对数据需求的任务调度方案.Fan 等人提出的 DALM 机制利用聚类算法将相互间关联性较高的数据进行分类,并实现其在集群间的公平分配[7];Shang 等人提出的 DRAW 机制利用数学方法得到表示数据间关联性程度的矩阵,再根据该矩阵完成关联性数据的在集群中的均衡分配[8].这两种机制都是基于集群同构的假设进行设计的,可在同构集群中实现负载均衡;但以上机制并未考虑异构集群中节点间计算、存储资源分布不均这一问题,在异构集群中可能导致资源利用率下降.

此外,减少 Shuffle 阶段不必要的数据迁移开销,也是研究人员关注的焦点.Pre-shuffling 机制[9]在数据经 Map 任务处理之前,由预测器根据数据迁移量最小这一原则做出预判,决定将数据传输至哪个节点进行 Reduce 任务处理.与其类似的解决方案还有文献[10,11]中提出的方案.与本文提出的解决方案不同,以上研究成果均未关注任务所处理数据间的依赖关系,而本文的任务优化调度机制则是建立在数据关联性的基础上进行任务调度决策的.

虽然当前关于大数据处理中任务调度优化策略的研究较多,但针对该问题的研究存在一定的局限性,因此

难以有效解决跨机架间不必要的数据迁移以及任务执行阶段较大的数据迁移量.本文设计的任务优化调度方案充分考虑了输入数据间的关联性以及数据本地性,建立数据、任务及节点三者间的对应关系,以数据迁移代价最小化为性能提升目标,设计优化数据放置策略及调度机制,提升大数据处理效率.

2 大数据处理任务最优调度问题描述与建模

在大数据处理中,若不考虑任务对数据的需求关系、节点对数据的放置状态、节点对任务的执行环境支撑关系,而在数据中心(data center,简称 DC)中随机放置数据或随意调度任务,则在任务执行过程中就可能产生大量跨节点、跨机架甚至跨 DC 的数据迁移,影响任务执行性能.然而与节点内的 I/O 相比,机架间和 DC 间的互连带宽通常较小,因此,如何减少任务执行过程的大量不必要的数据迁移,则是降低带宽使用代价、提升任务执行性能的关键.

假设 DC 中有 R 个机架,其中,机架 $r(r\in[1,R])$内有 N_r 个节点,这些节点可表示为 n_{i_r} ,其中,$r_i\in[1,N_r]$,DC 中所有节点集合为 N,共有$|N|(|N|=\sum_{r=1}^{R}N_r)$个节点.DC 内待调度的任务集合为 T,任务数为 J,任务 $t_j(j\in[1,J])$所需的输入数据集为 $D^{t_j}=\{d_l^{t_j}\mid l\in[1,L^{t_j}]\}$ (其中, L^{t_j} 为 D^{t_j} 包含的数据块数量),则该 DC 内的数据集合为 $D=\bigcup_{j=1}^{J}D^{t_j}$.

DC 中节点、任务、数据三者间的关系如图 1 所示.该图可看做两个二分图的组合:上半部的二分图 $G_1=(T\cup N,S)$表示任务与节点间的关系,其中,边的集合 $S=\{s_j|j\in[1,J]\}$代表调度决策;下半部的二分图 $G_2=(N\cup D,P)$则表示节点与待处理数据间的关系,其中,边的集合 $P=\{p_j|j\in[1,J]\}$代表数据放置决策.图 1 给出了 3 种有代表性的数据放置和任务调度情形,即:任务执行节点与数据放置节点的关系为同一节点(例如数据集 D^{t_j} 和任务 t_j)、同一机架(例如数据集 D^{t_1} 和任务 t_1)或不同机架(例如:数据集 D^{t_J} 和任务 t_J).不同情形下,任务执行所需的数据迁移量差异较大.前两者满足数据本地性(即,任务执行节点上或它所属的机架上是否有该任务所需的数据)和数据关联性(即,同一任务所需的数据是否都放置在同一节点或同一机架内不同节点上)要求,它们的数据迁移较少,因此,常常是任务调度和数据放置追求的目标情形.

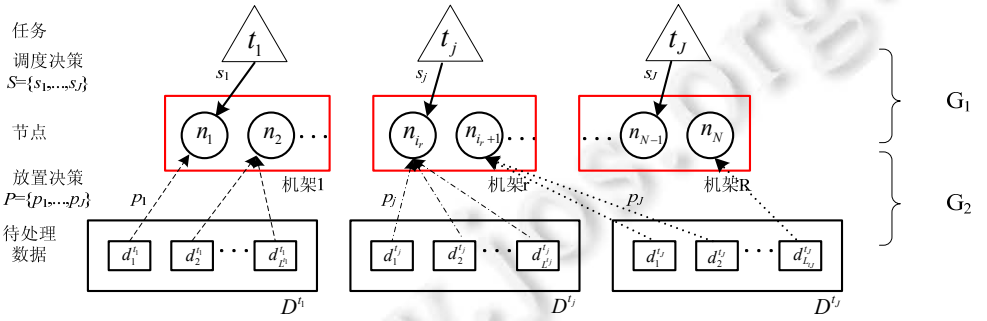


Fig.1 Scheduling and placement relationship among tasks, nodes and data

图 1 任务、节点与数据的调度与放置关系

通常,DC 中 J 个任务的总执行代价 TC 由计算代价 c_c 和数据迁移代价 c_t 两部分组成,即:

$$TC=c_c+c_t \tag{1}$$

其中,c_c 直接取决于各任务 t_j 计算时间 T_{exec} 和队列等待时间 T_{wait} 之和,即:

$$c_c(t_j)=\alpha_c\cdot(T_{exec}^{t_j}+T_{wait}^{t_j}) \tag{2}$$

其中,α_c 表示单位时间内使用计算资源的代价(费用).

而考虑到各任务的执行截止时间 $TH_{deadline}^{t_j}$ 要求,它们需满足 $T_{exec}^{t_j}+T_{wait}^{t_j}\leqslant TH_{deadline}^{t_j}$.因各节点的计算能力和存储数据各异,对这 J 个任务实施不同的调度决策 S 会影响其任务执行的计算代价,因此,c_c 可表示为任务调度方

案 S 的函数,即:

$$c_c=\sum_{t_j}c_c^{t_j}=z_1(S) \tag{3}$$

而 c_t 则与数据迁移量及迁移带宽有关,它们决定了数据迁移时间 T_{trans}.考虑到任务的执行截止时间要求,需使 $T_{trans}^{t_j}\leqslant TH_{deadline}^{t_j}$.如图 1 所示,对于$\forall t_j$,其执行节点为 n_{t_j},其输入数据 $d_1^{t_j},d_2^{t_j},...,d_{L^{t_j}}^{t_j}$ 放置于 $n_{i_{r1}},...,n_{i_{rL}}$,则 t_j 的数据迁移代价可表示为

$$c_t(t_j)=\alpha_t\sum_{d^{t_j}}T_{trans}^{t_j}=\alpha_t\sum_{a=1}^{L^{t_j}}\frac{|d_a^{t_j}|}{BW(n_{i_{ra}},n_{t_j})} \tag{4}$$

其中,α_t 表示单位时间内使用带宽资源的代价(费用), $BW(n_{i_{ra}},n_{t_j})$ 表示任务执行节点 n_{t_j} 与数据放置节点 $n_{i_{ra}}$ 间的互连带宽.

因此,J 个任务的总传输代价 $c_t=\sum_{j=1}^{J}c_t(t_j)$.实际上,数据放置策略 P 的调整会影响数据的本地性和数据的关联性[12],从而导致 c_t 变化.另外,由公式(4)可知:对于$\forall t_j$,其执行节点的调整也会类似地影响 c_t 的大小,即,c_t 会受到 S 的影响.因此,c_t 可以表示为

$$c_t=z_2(P,S) \tag{5}$$

综上所述,TC 可表示为

$$TC=c_c+c_t=z_1(S)+z_2(P,S) \tag{6}$$

由图 1 可知:若 n 时刻改变 P_n,则需对 $n-1$ 时刻的 S_{n-1} 优化调整为 S_n,才能满足数据本地性要求,因此,S_n 是随 P_n 而变,用函数 $p()$可表示如下:

$$S_n=p(P_n) \tag{7}$$

同理,若 n 时刻 S_n 改变,则需对该时刻的 P_n 进行优化调整,生成 P_{n+1} 以适应新调度方案的数据本地性要求,因此,P_{n+1} 是随 S_n 而变,用函数 $s()$可表示如下:

$$P_{n+1}=s(S_n) \tag{8}$$

大数据处理任务最优调度问题可表示为在数据本地性和数据关联性约束下的非线性优化问题,其优化目标为总执行代价 TC 最小化,即:

$$\begin{aligned}&\text{Min } TC_n=\text{Min } z_1(S_n)+z_2(P_n,S_n)\\&\text{s.t. }\begin{cases}S_n=p(P_n)\\P_{n+1}=s(S_n)\\T_{exec}^{t_j}+T_{wait}^{t_j}\leqslant TH_{deadline}^{t_j},\forall t_j\\T_{trans}^{t_j}\leqslant TH_{deadline}^{t_j},\forall t_j\end{cases}\end{aligned} \tag{9}$$

当 TC 取最小值时,所对应的调度方案 S 及数据放置策略 P 即为调度问题的最优解.根据公式(7)及公式(8),求解最优调度方案的过程即对 S 和 P 不断迭代调整的过程,其中,P 向 S 提供数据关联性信息作为决策依据,S 向 P 提供数据本地性信息作为决策依据,两者相互影响,直至达到 TC 的收敛.

3 D^{3}S^2:数据关联性驱动的任务最优调度方案

如何充分利用数据间关联性,尽可能地将同一任务所需的数据放置在同一节点或同一机架,以减少任务执行过程中的数据迁移,减少任务完成总时间,是提升任务执行性能的关键.为此,基于任务优化调度模型,本节提出了 D^{3}S^2 任务最优调度方案,其核心思想为:以数据关联性为约束,优化数据放置,以减少任务执行过程中机架间的数据迁移;以数据本地性为约束,优化任务调度,以减少任务执行过程中机架内的数据迁移.因此,该方案由数据优化放置(dependency-aware placement mechanism,简称 DAPM)和任务优化调度(transfer-aware scheduling mechanism,简称 TASM)两个核心机制组成.

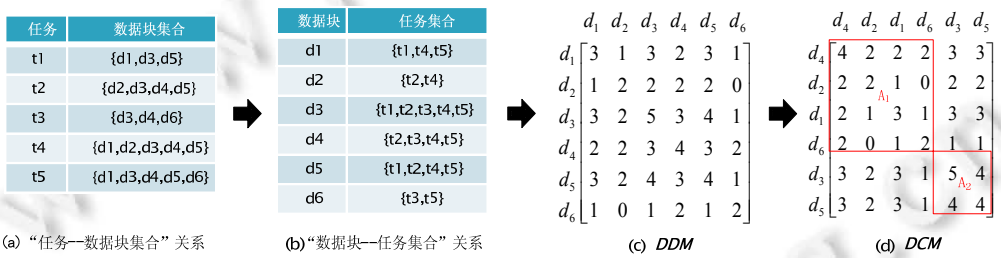
3.1 DAPM:数据关联性感知的数据优化放置机制

在 DC 内,当两个数据被很多任务频繁地同时访问时,可称这两个数据为强关联数据.当两个数据被更多地任务同时访问,则它们之间的关联程度越高,因此,可用任意两个数据块的任务集合取交集衡量这两个数据块间的关联性.若需要访问数据块 d_a 的任务集合定义为 T_a,d_b 的任务集合为 T_b,则数据块 d_a 与 d_b 间的关联性大小可以定义为 $D_{a,b}=||T_a \cap T_b||$.例如:图 2(b)所示 d_1 的任务集合 $T_1=\{t_1,t_4,t_5\}$,d_2 的任务集合 $T_2=\{t_2,t_4\}$,则 $D_{1,2}=||T_1 \cap T_2||=||\{t_4\}||=1$.类似地,$D_{1,3}=3$.由此可得数据集合 $D=\bigcup_{j=1}^{J} D^{t_j}$ 的关联性矩阵 ***DDM***,其中,任意元素由 $D_{a,b}$ 定义而得.图 2 给出了如何得到 ***DDM***:首先,从日志文件中获得如图 2(a)所示的“单个任务~数据块集合”关系图;然后再转换为“单个数据块~任务集合”关系图;最后,根据定义得到图 2(c)所示的关联性矩阵 ***DDM***.

DDM 仅反映了任意两个数据块间的关联性大小,然而该矩阵没有给出哪些数据块可以一起放置到同一个节点或机架.为了挖掘以 $m(m>2)$个数据块为单位的数据块簇间的关联性大小,这里需对矩阵 ***DDM*** 进行行列变换,并按公式(10)计算变换后矩阵的键能 B[13],直至找到最大键能值 B 对应的变换矩阵,即,数据聚合矩阵 ***DCM***.

$$B=\sum_{i=1}^{N}\sum_{j=1}^{N} D_{i,j}[D_{i,j-1}+D_{i,j+1}+D_{i-1,j}+D_{i+1,j}] \tag{10}$$

对图 2(c)所示的 ***DDM*** 进行逐次行变换,可得到如图 2(d)所示的最大键能值对应的变换后矩阵 ***DCM***.在 ***DCM*** 中,强关联的元素聚集在一起,其中,子矩阵 A_1,A_2 分别为具有 4 个和 2 个强关联数据块的簇.



(a) “任务--数据块集合” 关系 (b)“数据块--任务集合” 关系 (c) *DDM* (d) *DCM*

Fig.2 Steps of DAPM mechanism

图 2 DAPM 机制的步骤

然而,如何将尽可能大的强关联数据块簇放置在同一个机架,以最大程度地减少机架间数据迁移[14]?这里提出了一种基于 ***DCM*** 最大维子矩阵分割的数据优化放置机制,其基本思路是:以机架或节点当前最大可用存储容量为约束,确定子矩阵的最大分割维数.它的实现具体是:首先,对 DC 内所有机架当前的可用存储容量降序排列,若某机架 r 的当前可用存储容量 M_r 最大(即,最多可存储 M_r 个数据块),则在 ***DCM*** 中寻找最大分割维数为 M_r 的子矩阵 A_r,由此,数据块簇 A_r 可放置于该机架;然后类似地,对该机架 r 内各节点的当前可用存储容量降序排列,若某节点 n_{i_r} 的当前可用存储容量 M_{i_r} 最大,则在 A_r 中再寻找最大分割维数为 M_{i_r} 的子矩阵 A_{i_r},并将对应数据块簇放置于节点 n_{i_r}.

综上所述,DAPM 数据优化放置机制的主要思路见算法 1.根据文献[8]得知,将 ***DDM*** 聚类为 ***DCM*** 的过程是 NP-hard 问题,则得到最优 DCM 的时间复杂度为 $O(n^n)$(其中,n 为矩阵维度);而本文使用的得到 ***DCM*** 的算法将时间复杂度节省至 $O(n^2)$[13];另外,对于 ***DCM*** 矩阵的分割及数据放置,其时间复杂度为 $O(R\times|N|)$.由此可知,DAPM 数据优化放置机制的时间复杂度较为合理.

需要额外说明的是:随着数据中心内数据的不断更新,其计算任务与数据块间的关联性也会随之更新;当关联性关系改变而引起原有数据放置决策的数据迁移代价显著增大时,则会再次触发数据优化放置机制,得到当前关联性信息下的最优解.

算法 1. DAPM 数据优化放置机制.

1) Input: **LOGS**, R racks, N nodes;

2) Output: 数据放置方案 *P*.
3) **when** (**LOGS** 读取完毕) **do**
4) (*block_id*, *task_id*) ***DDM***
5) **when** (Bond energy 达到最大值) **do**
6) ***DDM***, ***DCM***
7) **do while** (***DCM*** 没有分割完毕)
8) 对 *R* 个机架按其最大可用存储容量 M_r 进行降序排列,记为 $M_1>\ldots>M_r>\ldots>M_R$
9) **for** *r* from 1 to *R* **do**
10) 从 ***DCM*** 上分割 M_r 维的子矩阵 $\boldsymbol{DCM_r}$
11) **do while** (子矩阵 $\boldsymbol{DCM_r}$ 没有分割完毕)
12) 对机架 *r* 内的节点按其最大可用存储容量 M_{ir} 进行降序排列,记为 $M_1>\ldots>M_{ir}>\ldots>M_{Nr}$
13) **for** $\boldsymbol{i_r}$ **from 1 to** $\boldsymbol{N_r}$ **do**
14) 从子矩阵 $\boldsymbol{DCM_r}$ 分割 M_{ir} 的子矩阵 $\boldsymbol{DCM_{ir}}$
15) **endfor**
16) **endfor**

3.2 TASM:数据迁移代价感知的任务优化调度机制

本节假设 DC 内计算资源相对较为充足,调度策略的调整不会给计算代价 c_c 带来明显变化,因此,最小化 *TC* 等效为最小化 c_t.此外,考虑到一般情况下,跨机架的节点间带宽通常小于机架内节点间带宽,因此,这里假设 $BW_{inter}<<BW_{inside}$.

假设 *n* 时刻的数据放置方案为 P_n,DC 内任意任务 $t_j(j\in[1,J])$的输入数据集 $D^{t_j}=\{d_l^{t_j}\mid l\in[1,L^{t_j}]\}$ 存放于节点集合 $N^{t_j}=\{n_l^{d_{t_j}}\mid l\in[1,L^{t_j}]\}$.当前任务调度方案 S_n 可表示为任务~执行节点关系 $t_j\sim n_{t_j}$ 的集合,即:

$$S_n=\{s_{t_j}:t_j\sim n_{t_j}\mid j\in[1,J]\}.$$

根据任务执行节点 n_{t_j} 与数据放置节点 $n_l^{d_{t_j}}\in N^{t_j}$ 的关系,可将公式(3)的 t_j 数据迁移代价 $c_t(t_j)$表示为跨机架数据迁移分量 $c_t(t_j)_{inter\text{-}rack}$ 与机架内数据迁移分量 $c_t(t_j)_{in\text{-}rack}$ 之和,即:

$$c_t(t_j)=c_t(t_j)_{inter\text{-}rack}+c_t(t_j)_{in\text{-}rack} \tag{11}$$

为达到数据迁移代价最小的优化调度目标,结合 $BW_{inter}<<BW_{inside}$ 的约束,在任务调度时,需尽可能使任务执行节点 n_{t_j} 与数据放置节点 $n_l^{d_{t_j}}$ 为同一节点或处于同一机架上,以减少数据迁移量[15],尤其是开销更大的跨机架数据迁移分量 $c_t(t_j)_{inter\text{-}rack}$.这里定义指引函数 $i(n_l^{d_{t_j}},n_{t_j})$ 表示任务执行节点 n_{t_j} 与数据放置节点 $n_l^{d_{t_j}}\in N^{t_j}$ 的关系,即:

$$i(n_l^{d_{t_j}},n_{t_j})=\begin{cases}1, & n_l^{d_{t_j}}=n_{t_j}\\ 0, & n_l^{d_{t_j}}\in N_{r_1},n_{t_j}\in N_{r_2},N_{r_1}=N_{r_2}\\ -1, & n_l^{d_{t_j}}\in N_{r_1},n_{t_j}\in N_{r_2},N_{r_1}\cap N_{r_2}=\varnothing\end{cases},\forall n_l^{d_{t_j}}\in N^{t_j} \tag{12}$$

若用 $I(n_{t_j})=\sum_{l=1}^{L^{t_j}}i(n_l^{d_{t_j}},n_{t_j})$ 表示各任务调度策略 s_{t_j} 的数据本地性程度,则最小化数据迁移代价 $c_t(t_j)$可等效为最大化 $I(n_{t_j})$.当 $I(n_{t_j})$ 取最大值时,此时对应的调度决策 s_{t_j} 为最优.因此,当 $I(S_n,P_n)=\sum_{j=1}^{J}I(n_{t_j})$ 取最大值时,对应的 $S_n=\{s_{t_j}\mid j\in[1,J]\}$ 即为在当前数据放置方案 P_n 约束下,DC 内 *J* 个任务的最优调度决策.由此可进一步描述数据迁移代价感知的任务调度最优化问题.

$$\text{Min } c_t = \text{Min}\sum_{j=1}^{J} c_t(t_j) \text{等价于} \text{Max } I(S_n,P_n) = \text{Max}\sum_{j=1}^{J} I(n_{t_j})$$

$$\text{s.t.} \begin{cases} S_n = p(P_n) \\ P_{n+1} = s(S_n) \\ c_t(t_j)_{\text{int}er\text{-}rack} << c_t(t_j)_{in\text{-}rack}, \forall t_j \\ T_{trans}^{t_j} \leqslant TH_{deadline}^{t_j}, \forall t_j \end{cases} \tag{13}$$

对于 DC 中的 J 个任务,理想情况下,任意任务 t_j 的输入数据均恰好放置于其执行节点 n_{t_j},此时,$I(S_n,P_n)$取最大值 $I_{ideal}(S,P)$,则 n 时刻调度方案 S_n 的数据本地性程度β_n 定义为 $\beta_n = \dfrac{I(S_n,P_n)}{I_{ideal}(S,P)}$.在 S_n 最优调度方案下,“任务~数据块”关系不同于 n–1 时刻的.上式的约束 P_n 是否需要在新的数据关联性下采用第 4.1 节的机制进一步优化?这取决于β_n 与β_{n-1} 的关系.若$\beta_n \geqslant \beta_{n-1}$,则可以进一步优化放置方案 P_n 为 P_{n+1}.同时,当 $c_{t_n} \leqslant c_{t_{n-1}}$ 时,可以在当前数据本地性约束下进一步优化调度方案 S_{n-1} 为 S_n.而当 $c_{t_{n-1}} - c_{t_n} < \delta$ (δ为一很小的正实数)时,此时对应的 P_n,S_n 即为最优调度方案.

进一步讨论计算代价 c_c 可变的情况,即同时考虑 c_c 和 c_t 对于 TC 的影响.此时,当 $c_{t_n} \leqslant c_{t_{n-1}}$,即可继续迭代调整.这是因为当$\beta_n<\beta_{n-1}$ 时,可能此时计算资源分配更为合理,使得 c_c 有效减少,同样使得 $TC_n \leqslant TC_{n-1}$.

3.3 D^3S^2任务优化调度方案实现

本文设计的 D^3S^2 优化调度方案结构如图 3 所示,主要由两部分组成:(1) 数据放置引擎,主要完成数据关联性发现及数据放置工作;(2) 任务调度引擎,完成数据迁移代价最小化的任务统一调度工作.另外,图 3 中 DC 内的大数据处理框架可选用 Hadoop/Spark 等,而文件系统则为 DC 处理框架对应的分布式文件系统,如 HDFS 等.日志仓库中存有任务执行过程所产生的历史日志,包括任务 ID、数据 ID、节点 ID、各数据块大小、数据流向信息、任务执行进度等.在 D^3S^2 中,数据放置引擎可通过解析历史日志得到帮助数据放置及任务调度决策的关键信息,如:根据任务 ID 及数据 ID 得到各任务与其所处理数据块间的对应关系,进而由数据放置引擎根据 DAPM 机制得到数据块间的关联关系;根据数据流向信息、节点 ID 及各数据块大小得到执行过程中的数据迁移量统计信息等.该任务调度优化方案可作为相应大数据处理框架的可选调度方案集成至相应框架内,如作为 Hadoop 中的可插拔调度器.

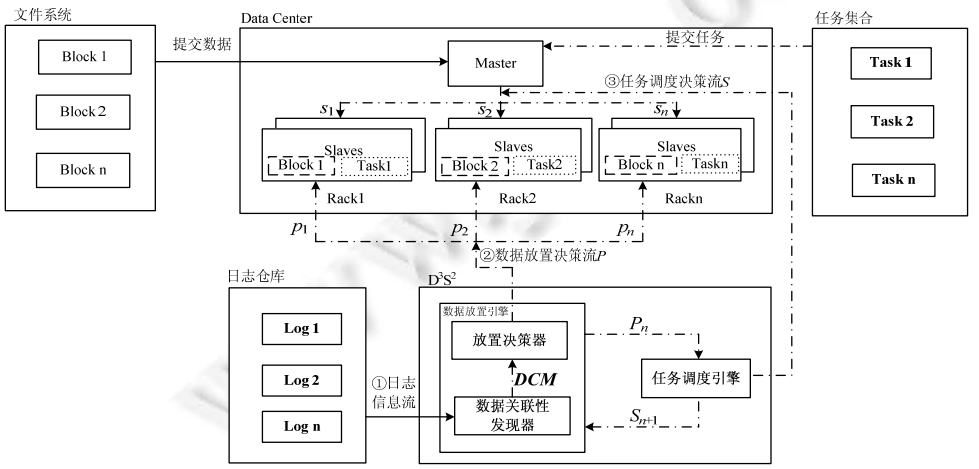


Fig.3 Diagram of D^3S^2's architecture and information flow

图 3 D^3S^2 结构及信息流向示意图

此外,图 3 反映了 D^3S^2 工作过程及信息流向情况,图中实线代表实际数据流(日志、数据等),虚线代表各决

策信息流,其工作过程可描述为:① 数据放置引擎提取历史日志进行数据关联性分析,并将当前数据关联性信息提供给任务调度引擎作为调度决策依据,同时,任务调度引擎将当前数据本地性信息提供给数据放置引擎作为数据放置决策依据;② *TC* 达到收敛后,数据放置引擎向文件系统传送数据放置策略,完成数据放置;③ 任务调度引擎向数据中心提供执行代价最小化的优化调度方案决策,并由 Master 完成任务调度.

4 方案验证与性能分析

这里,我们搭建了一个 Hadoop 集群环境来分析 D^3S^2 的性能:首先,针对 DAPM 优化放置机制及 TASM 优化调度机制分别进行了性能分析;之后,将两种优化手段结合起来,对 D^3S^2 进行了总体性能分析.

4.1 实验环境设置

本文采用在 Windows 服务器下构建虚拟机的方式搭建了如图 4 所示的数据中心中较为普遍的 Hadoop(版本为 1.2.1)大数据分析环境,包括 12 个节点(各节点配置情况见表 1),分别放置于 3 个机架中.

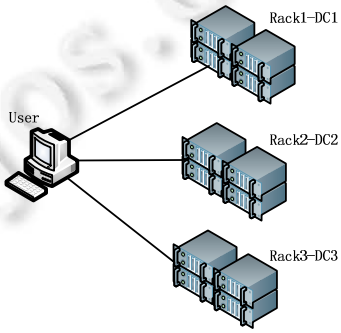


Fig.4 Diagram of the experiment environment

图 4 实验环境示意图

Table 1 Configurations of experiment environment

表 1 实验环境配置

	Host1(Master)	Host2-4	Host5-6	Host7-8	Host9-10	Host11-12
所属机架	Rack 1	Rack 1	Rack 2	Rack 2	Rack 3	Rack 3
内存	4G	1G	1G	1G	512M	512M
硬盘	20G	20G	10G	10G	10G	10G

本文实验的任务实例都来自于 Intel 的 Hibench[16]套件.为保证 D^3S^2 测试有效性,我们选用了 HiBench 中 PageRank[17]和 Terasort 两类典型作业,其中,前者体现网页数据之间相关性,后者 Shuffle 阶段存在大量数据迁移.所用测试数据集大小为 5GB~60GB,为了使数据块数量达到一定规模,增加分析环境中数据块数量,达到充分模拟数据块间关联性的目的,数据块大小设置为 32MB,副本数为 1.

以上实验环境的配置充分模拟了实际数据中心的典型特征,同时也保留了数据中心的处理模式.因此,利用该数据中心模拟环境进行实验验证具有可行性及可信性.

4.2 DAPM优化放置机制性能分析

本节共设 4 组对照实验,分别为:(1) 原生 Hadoop;(2) 配置了 Fair 调度器的 Hadoop;(3) MatchMaking[18],未对关联性数据进行处理的数据放置机制;(4) DAPM 机制.这里采用 PageRank 进行实验,首先验证 DAPM 机制对于减少跨机架数据迁移的能力有无,之后考察 DAPM 对于跨机架数据迁移的减少程度,以及该机制对于执行速度的提升.

图 5 反映了 DAPM 和原生 Hadoop 的对于跨机架数据迁移量的影响(这里设置输入数据大小为 30G,选择 Update Rank 阶段的数据迁移进行分析).从图中可知:在执行相同作业的条件下,DAPM 的跨机架数据迁移随日

志迭代分析过程不断减少,而原生 Hadoop 的跨机架数据迁移则在反复迭代过程中未有明显减少的趋势,证明了 DAPM 机制对于数据关联性的挖掘及聚合放置有效减少了任务执行过程中不必要的数据迁移.此外,观察 DAPM 的跨机架迁移变化曲线:第 1 次迭代时,数据迁移量下降率较小,这是由于初始时任务-数据块关系随机,数据间关联性信息较为有限;随着迭代次数的增加,数据迁移量下降较为明显;至迭代 6~7 次时,数据迁移量趋于收敛,这时,输入数据间的关联性被较为完整地挖掘出来.此时,数据迁移代价达到相对收敛,应停止迭代;如图 5 所示,第 7 次迭代后,继续迭代的跨机架数据迁移减小不明显,此时,对于计算资源的消耗占主导,相对迭代代价(任务执行总代价)将逐渐增加.另外,与原生 Hadoop 相似,Fair 调度机制及 MatchMaking 机制均未针对数据关联性进行处理,因此,这里仅对 DAPM 与原生 Hadoop 进行比较.

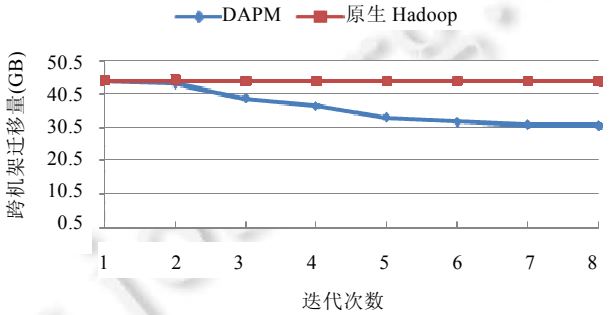
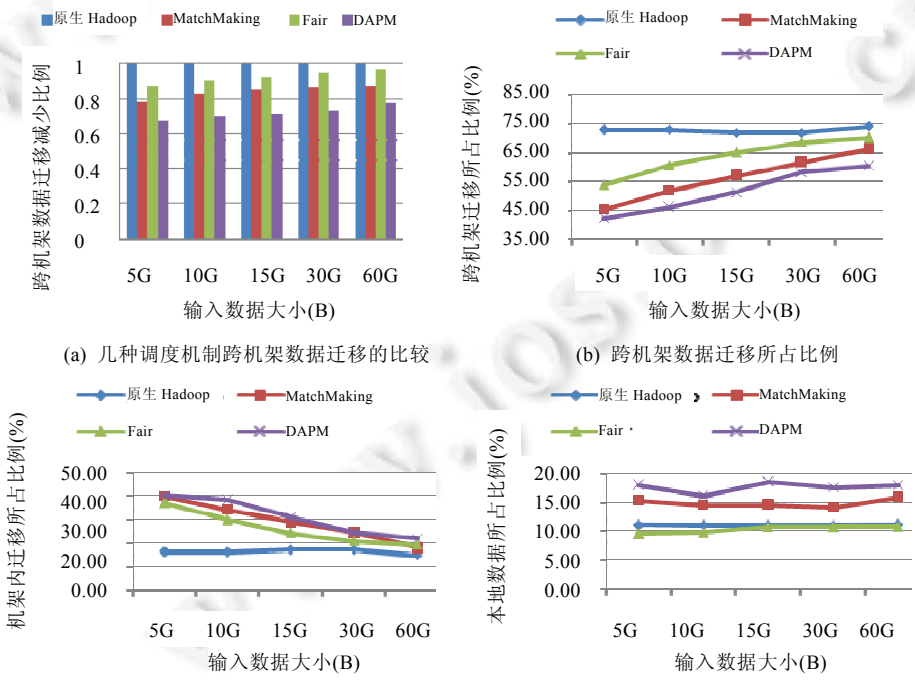


Fig.5 Effect of iterations on cross-rack data transfer

图 5 迭代次数对跨机架数据迁移的影响

图 6 进一步反映了 DAPM 机制对于跨机架数据迁移量的减少程度.



(a) 几种调度机制跨机架数据迁移的比较

(b) 跨机架数据迁移所占比例

(c) 机架内数据迁移所占比例

(d) 本地数据所占比例

Fig.6 Effect of DAPM on data transfer and its ratio

图 6 DAPM 优化放置机制对数据迁移及其比例的影响

由图 6(a)所示可以看出,

- 在不同规模的输入数据下,DAPM 的跨机架数据迁移量相对于其他调度机制有 22%~33%的减少;
- 相对于原生 Hadoop,DAPM 机制的跨机架数据迁移量平均下降了 17.3%.

进一步观察图 6(a)可发现:随着输入数据集的增大,跨机架迁移所占比例逐渐上升.主要原因为:在非理想状态下,由于节点存储容量限制而导致强关联数据分散放置于不同机架的节点上,且输入数据量越大,被分散放置的强关联性数据越多.图 6(b)~图 6(d)则反映了各类数据迁移在总数据迁移中所占比例.不难看出:在不同大小的输入数据作用下,相对于其他调度机制,DAPM 的跨机架迁移比例较少,而机架内数据迁移及本地数据比例较多.这是由于 DAPM 将强关联的数据放置于同一机架内,则原跨机架的数据迁移转化为机架内数据迁移或本地数据.

图 7 则反映了 DAPM 机制对于作业完成时间的影响.从图中信息可知:若不考虑迭代之后最优解,仅从单次执行角度出发,该机制相对于原生 Hadoop 平均有 17.3%左右的降低.

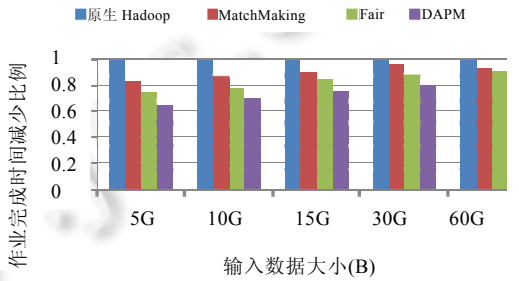


Fig.7 Effect of DAPM on job completion time

图 7 DAPM 优化放置机制对作业完成时间的影响

4.3 TASM任务优化调度机制性能分析

实验采用 TeraSort 对 TASM 优化调度机制进行性能分析,共设置 3 组对照实验,分别为:

(1) 原生 Hadoop;

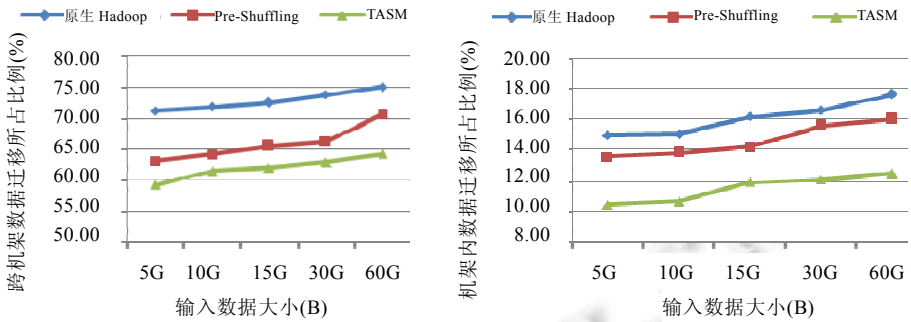
(2) Pre-Shuffling[9];

(3) TASM 机制.

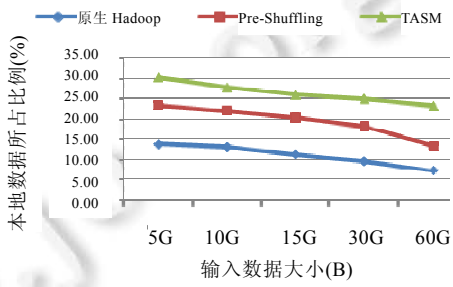
本节实验以 DAPM 机制所得优化放置决策为统一数据放置方案,进而分析 TASM 任务优化调度机制的性能.为准确分析 TASM 性能优势,首先对作业执行过程中各类数据迁移所占比例进行对比分析,并进一步比较其数据迁移代价;之后,通过对比作业完成时间分析 TASM 机制执行开销.

图 8 反映了 TASM 机制对于作业执行过程中的各类数据迁移的影响.如图 8(c)所示:在不同大小的输入数据环境下,TASM 机制的本地数据比例相对于原生 Hadoop 和 Pre-Shuffling 机制分别平均提高了 15.6%和 7.1%,而跨机架数据迁移及同机架数据迁移比例小于其他调度机制(如图 8(a)、图 8(b)所示).这是由于 TASM 机制将任务调度至其较多输入数据所存放的节点或机架上,且以数据迁移代价 $c_t=\sum c_t(t_j)$ 最小化为目标优化调度方案,由于调度方案的调整,原有的跨机架及机架内数据迁移一部分转化为本地数据,系统的数据本地性程度提高.如图 9 所示,TASM 机制的数据迁移代价相比原生 Hadoop 及 Pre-Shuffling 机制分别平均下降了 26.8%及 13.6%.

如图 10 所示:在不同的输入数据负载大小下,TASM 机制在作业完成时间上相对于原生 Hadoop 减少了 12.4%.这主要是因为 TASM 对于任务进行了统一调度,相对于其他机制分阶段调度任务的行为,节省了一部分的任务调度时间.



(a) 跨机架数据迁移所占比例 (b) 机架内数据迁移所占比例



(c) 本地数据所占比例

Fig.8 Effect of TASM on data transfer ratio

图 8 TASM 机制对于各类数据迁移所占比例的影响

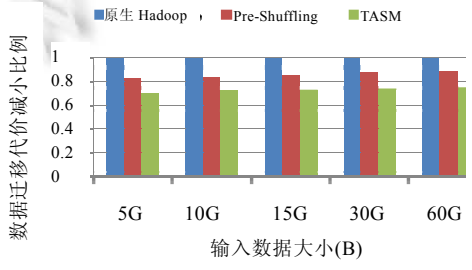


Fig.9 Effect of TASM on data transfer cost

图 9 TASM 机制对数据迁移代价的影响

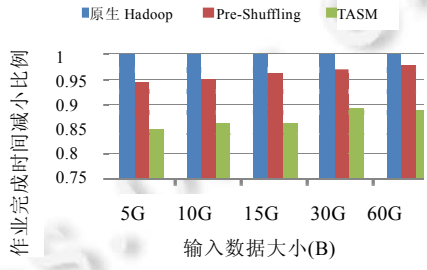


Fig.10 Effect of TASM on job completion time

图 10 TASM 机制对作业完成时间的影响

4.4 D^3S^2性能分析

D^3S^2 的性能分析选择见表 2 所示的综合实验集,在实验集设定上,分别侧重了数据关联性较强及数据迁移量较大两方面的因素,并将实验结果与原生 Hadoop 环境下运行的实验结果进行对比分析.

Table 2 Experiment set configuration

表 2 实验集设定情况

工作集	作业数	输入数据量(GB)
WorkSet 1	100 PageRank Jobs	30
WorkSet 2	100 TeraSort Jobs	30
WorkSet 3	100 PageRank Jobs, 100 TeraSort Jobs	60
WorkSet 4	50 PageRank Jobs, 150 TeraSort Jobs	60
WorkSet 5	150 PageRank Jobs, 50 TeraSort Jobs	60

在实验指标上,选择公式(1)中定义的执行总代价 TC 和数据迁移代价 c_t,并考量 D^3S^2 相对原生 Hadoop 的执

行速度提升.定义 *SpeedUp* 表示 D^3S^2 相对原生 Hadoop 的执行速度提升程度,即:

$$SpeedUp=\frac{\frac{1}{t_{optimized}}}{\frac{1}{t_{native}}}=\frac{t_{native}}{t_{optimized}} \tag{14}$$

图 11 所示为 D^3S^2 与原生 Hadoop 在作业执行速度上的对比.在不同实验集下,D^3S^2 在作业执行速度上相对于原生 Hadoop 有 9.4%~17.1%的提升.当输入数据大小为 30G 时,WordSet 1 的执行速度略高于 WordSet 2 的执行速度;而当输入数据大小为 60G 时,WordSet 5 的执行速度更快.这是由于 TeraSort 相对于 PageRank 数据迁移量更大,且数据间关联性较弱,因此对于 TeraSort 的数据放置和任务调度优化效果,相对于 PageRank 较为逊色.另外,从图 11 中可以看出:当输入数据增大,即 Job 数量增加时,性能提升效果更为明显.其主要原因为:原生 Hadoop 由于未针对资源占用情况以及数据间相关性对任务调度决策进行动态调整,因而在数据迁移以及任务队列等待上开销更大,而 D^3S^2 则弥补了这些方面的不足.

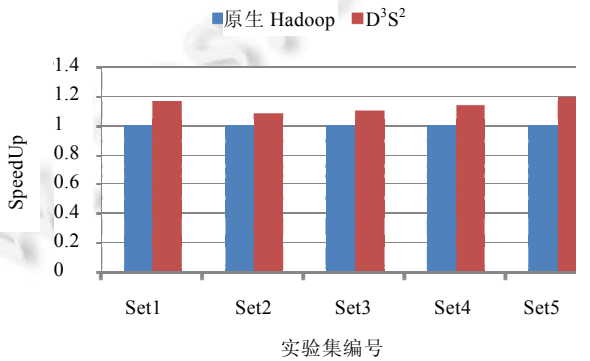


Fig.11 Performance comparison of D^3S^2 and native Hadoop on job execution speed

图 11 D^3S^2 与原生 Hadoop 在作业执行速度上的性能比较

图 12 所示分别为 D^3S^2 与原生 Hadoop 在作业执行期间数据迁移代价 c_t 及执行总代价 *TC* 的对比.相对于原生 Hadoop,D^3S^2 的执行总代价 *TC* 减少了 20.9%~31.8%,而数据迁移代价 c_t 则减少了 19.9%~32.4%.这是由于 D^3S^2 针对数据关联性对输入数据进行了合理放置,有效减少了开销更大的跨机架数据迁移,因此数据迁移代价明显减少;又因以最大程度实现数据本地性为优化目标进行任务统一调度,进一步减少了节点间的数据迁移.

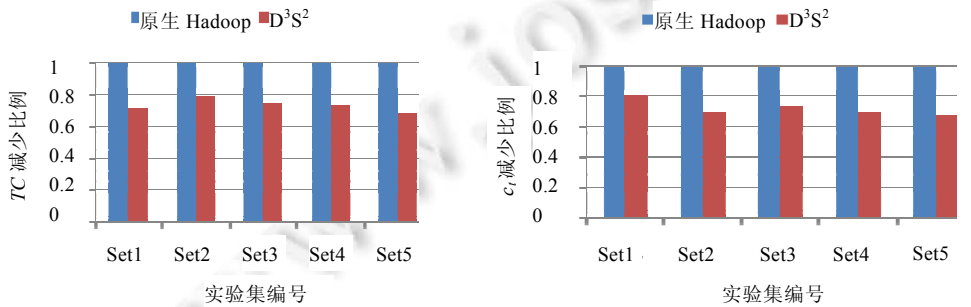


Fig.12 Performance comparison of D^3S^2 and native Hadoop on *TC* and c_t

图 12 D^3S^2 与原生 Hadoop 在 TC,c_t 上的性能比较

5 结束语

当前,MapReduce 等大数据处理框架由于忽视数据间相对任务的依赖关系,导致任务执行过程中产生大量的跨节点甚至跨机架数据迁移,影响任务执行效率.针对这种情况,本文通过对数据中心内任务调度问题的建模

分析,进而从数据关联性及数据本地性出发,提出了任务优化调度方案——D^3S^2.通过发现处理数据间的关联性,将强关联数据聚合放置,以减少跨机架数据迁移;并以数据迁移最小化为目标优化任务调度决策,进一步减少机架内数据迁移;数据放置决策与任务调度决策互相提供决策依据,不断迭代调整直至达到最优解.实验分析结果表明;D^3S^2 在降低作业执行时跨机架及跨节点的数据迁移方面具有较好的性能;同时,该方案的执行开销较小,不会影响作业的执行速度.

References:

[1] Cheng XQ, Jin XL, Wang YZ, Guo JF, Zhang TY, Li GJ. Survey on big data system and analytic technology. Ruan Jian Xue Bao/Journal of Software, 2014,25(9):1240−1252 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4674.htm [doi: 10.13328/j.cnki.jos.004674]

[2] Garcia-Dorado JL, Rao SG. Cost-Aware multi data-center bulk transfers in the cloud from a customer-side perspective. IEEE Trans. on Cloud Computing, 2015. [doi: 10.1109/TCC.2015.2469666]

[3] Xun YL, Zhang JF, Qin X. Data placement strategy for MapReduce cluster environment. Ruan Jian Xue Bao/Journal of Software, 2015,26(8):2056−2073 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4807.htm [doi: 10.13328/j.cnki.jos.004807]

[4] Guo D, Xie JJ, Zhou XL, Zhu XB, Wei W, Luo XS. Exploiting efficient and scalable shuffle transfers in future data center networks. IEEE Trans. on Parallel & Distributed Systems, 2014,26(4):997−1009. [doi: 10.1109/TPDS.2014.2316829]

[5] Gu T, Zuo C, Liao Q, Yang YL, Li T. Improving MapReduce performance by data prefetching in heterogeneous or shared environments. Int'l Journal of Grid & Distributed Computing, 2013,6(5). [doi: 10.14257/ijgdc.2013.6.5.07]

[6] Abad CL, Lu Y, Campbell RH. DARE: Adaptive data replication for efficient cluster scheduling. In: Proc. of the 2011 IEEE Int'l Conf. on Cluster Computing (CLUSTER). IEEE, 2011. 159−168. [doi: 10.1109/CLUSTER.2011.26]

[7] Fan XY, Ma XQ, Liu JC, Li D. Dependency-Aware data locality for MapReduce. In: Proc. of the 2014 IEEE 7th Int'l Conf. on Cloud Computing (CLOUD). IEEE, 2014. 408−415. [doi: 10.1109/TCC.2015.2511765]

[8] Shang PJ, Xiao QJ, Wang J. DRAW: A new data-grouping-aware data placement scheme for data intensive applications with interest locality. In: Proc. of the Cloud Computing for Data-Intensive Applications. New York: Springer-Verlag, 2014. 149−174. [doi: 10.1007/978-1-4939-1905-5_7]

[9] Seo S, Jang I, Woo K, Kim I, Kim JS, Maeng S. HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment. In: Proc. of the IEEE Int'l Conf. on Cluster Computing and Workshops (CLUSTER 2009). IEEE, 2009. 1−8. [doi: 10.1109/CLUSTR.2009.5289171]

[10] Hammoud M, Sakr MF. Locality-Aware reduce task scheduling for MapReduce. In: Proc. of the 2011 IEEE 3rd Int'l Conf. on Cloud Computing Technology and Science (CloudCom). IEEE, 2011. 570−576. [doi: 10.1109/CloudCom.2011.87]

[11] Hammoud M, Rehman MS, Sakr MF. Center-of-Gravity reduce task scheduling to lower mapreduce network traffic. In: Proc. of the 2012 IEEE 5th Int'l Conf. on Cloud Computing (CLOUD). IEEE, 2012. 49−58. [doi: 10.1109/CLOUD.2012.92]

[12] Wang MJ, Zhang JH, Dong F, Luo JZ. Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment. In: Proc. of the Int'l Conf. on Advanced Cloud & Big Data. IEEE, 2014. 77−84. [doi: 10.1109/CBD.2014.19]

[13] Gorla N, Zhang K. Deriving program physical structures using bond energy algorithm. In: Proc. of the 6th Asia Pacific Software Engineering Conf. (APSEC'99). IEEE, 1999. 359−366. [doi: 10.1109/APSEC.1999.809624]

[14] Zheng P, Cui LZ, Wang HY, Xu M. A data placement strategy for data-intensive applications in cloud. Chinese Journal of Computers, 2010,33(8):1472−1480 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2010.01472]

[15] Bhushan M, Yadav SK. Cost based model for big data processing with hadoop architecture. Global Journal of Computer Science and Technology, 2014,14(2).

[16] Huang SS, Huang J, Dai JQ, Xie T, Huang B. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: Proc. of the 2010 IEEE 26th Int'l Conf. on Data Engineering Workshops (ICDEW). IEEE, 2010. 41−51. [doi: 10.1109/ICDEW.2010.5452747]

[17] Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the Web. 1999.

[18] He C, Lu Y, Swanson D. Matchmaking: A new mapreduce scheduling technique. In: Proc. of the IEEE, Int'l Conf. on Cloud Computing Technology and Science (Cloudcom 2011). Athens, 2011. 40−47. [doi: 10.1109/CloudCom.2011.16]

附中文参考文献:

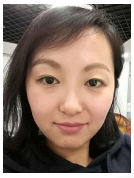
[1] 程学旗,靳小龙,王元卓,郭嘉丰,张铁赢,李国杰.大数据系统和分析技术综述.软件学报,2014,25(9):1889−1908. http://www.jos.org.cn/1000-9825/4674.htm [doi: 10.13328/j.cnki.jos.004674]

[3] 荀亚玲,张继福,秦啸.MapReduce 集群环境下的数据放置策略.软件学报,2015,26(8):2056−2073. http://www.jos.org.cn/1000-9825/4807.htm [doi: 10.13328/j.cnki.jos.004807]

[14] 郑湃,崔立真,王海洋,徐猛.云计算环境下面向数据密集型应用的数据布局策略与方法.计算机学报,2010,33(8):1472−1480. [doi: 10.3724/SP.J.1016.2010.01472]

王玢(1991－),女,河北石家庄人,硕士生,主要研究领域为大数据处理技术优化.

阳小龙(1970－),男,博士,教授,博士生导师,主要研究领域为新一代互联网理论与技术,NGB,物联网技术.



吴雅婧(1982－),女,博士,主要研究领域为大数据.

孙奇福(1983－),男,博士,副教授,主要研究领域为网络编码,编码理论.