软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(8):2091−2110 [doi: 10.13328/j.cnki.jos.004802]

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

一种优化 MapReduce 系统能耗的数据布局算法*

宋 杰[1], 王 智[1], 李甜甜[2], 于 戈[2]

[1](东北大学 软件学院,辽宁 沈阳 110819)

[2](东北大学 信息科学与工程学院,辽宁 沈阳 110819)

通讯作者: 宋杰, E-mail: songjie@mail.neu.edu.cn, http://www.neu.edu.cn

摘 要: 在云计算技术和大数据技术的推动下,IT 资源的规模不断扩大,其能耗问题日益显著.研究表明:节点资源利用率不高、资源空闲导致的能源浪费,是目前大规模分布式系统的主要问题之一.研究了 MapReduce 系统的能耗优化.传统的基于软件技术的能耗优化方法多采用负载集中和节点开关算法,但由于 MapReduce 任务的特点,集群节点不仅要完成运算,还需要存储数据,因此,传统方法难以应用到 MapReduce 集群.提出了良好的数据布局可以优化集群能耗.基于此,首先定义了数据布局的能耗优化目标,并提出相应的数据布局算法;接着,从理论上证明该算法能够实现数据布局的能耗优化目标;最后,在异构集群中部署 3 种数据布局不同的 MapReduce 系统,通过对比三者在执行 CPU 密集型、I/O 密集型和交互型这 3 种典型运算时的集群能耗,验证了所提出的数据布局算法的能耗优化效果.理论和实验结果均表明,所提出的布局算法能够有效地降低 MapReduce 集群的能耗.上述工作都将促进高能耗计算和大数据分析的应用.

关键词: 能耗优化;MapReduce;数据布局;大数据

中图法分类号: TP316

中文引用格式: 宋杰,王智,李甜甜,于戈.一种优化 MapReduce 系统能耗的数据布局算法.软件学报,2015,26(8):2091−2110. http://www.jos.org.cn/1000-9825/4802.htm

英文引用格式: Song J, Wang Z, Li TT, Yu G. Energy consumption optimization data placement algorithm for MapReduce system. Ruan Jian Xue Bao/Journal of Software, 2015,26(8):2091−2110 (in Chinese). http://www.jos.org.cn/1000-9825/4802.htm

Energy Consumption Optimization Data Placement Algorithm for MapReduce System

SONG Jie[1], WANG Zhi[1], LI Tian-Tian[2], YU Ge[2]

[1](Software College, Northeastern University, Shenyang 110819, China)

[2](School of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: Driven by big data and cloud computing techniques, the scale of the IT expenditure grows continuously and energy consumption problem has become more and more urgent. Study shows that the lower resource usage and the long idle time of network nodes are responsible for this problem in a large-scale distributed system. This paper studies the energy consumption optimization of MapReduce system. Traditional optimization approaches employ workload concentration, task live-immigration or dynamical power on-off methods. But in a MapReduce system, a node not only executes tasks but also provides data, therefore cannot be simply shut down for energy-saving while the tasks running on it are migrated. This paper presents an idea that a good data placement can optimize the energy consumption of a MapReduce system. Based on this idea, the target of data placement which optimizes the energy consumption is defined. Then the data placement algorithm achieving the target is proved efficient in theory. Finally, three MapReduce systems with different data placement algorithms are deployed on the heterogeneous MapReduce system. Comparing the energy consumption of three systems under the three typical CPU-intensive, I/O intensive and interactive jobs, the proposed data placement algorithm is proved to be

* 基金项目: 国家自然科学基金(61202088, 61433008); 中国博士后科学基金(2013M540232); 教育部高等学校博士学科点专项科研基金(2012004211 0028); 中央高校基本科研业务费种子基金(N130417001)

收稿时间: 2014-06-11; 修改时间: 2014-07-30; 定稿时间: 2014-12-09

able to optimize the energy consumption of a MapReduce system. The optimization efficiency of the proposed approach is proved both in theory and by experiment, demonstrating its ability to facilitate the applications of energy consumption computing and big data analysis.
Key words: energy consumption optimization; MapReduce; data placement; big data

近年来,随着信息化程度的日益加深,人们积累了海量数据[1].能否高效地管理和分析大数据,影响着信息化社会各行各业的发展.MapReduce[2]是处理大数据的一种有效的手段,但随着 IT 设施的功率逐渐增加和 MapReduce 分布式系统规模的不断扩大,系统能耗过高已经成为 MapReduce 应用领域中的一个难题.多伦多的一个非营利性环境组织发现,数据中心的绝大多数服务器的运行效率低于 4%[3].在此背景下,如何优化 MapReduce 系统能耗,成为当今研究的热点问题[4−13].能源可以视为由硬件系统直接消耗,也可以视为由软件系统间接消耗,若将 MapReduce 系统的软件和硬件部分视为一个整体,那么其能耗优化就是在任务量一定的前提下降低系统消耗的能量,或是在运算能力不变的前提下降低能耗.本研究着重从数据布局角度优化 MapReduce 系统的能耗.在 MapReduce 模型中,作业会被分解成若干种顺序执行的任务(如 Map 任务执行完毕后 Reduce 任务开始执行),同种任务(的实例)则部署到各个节点并行执行,任务的执行时间取决于最后执行结束的节点.与此同时,其他节点均会处于等待状态.我们的前期研究证明[4],数据密集型计算会导致节点资源使用率不高.产生这一情况的原因可以归结为节点资源等待,如等待网络数据传输或者等待其他节点,我们称因节点资源被动空闲而产生的能耗为等待能耗.提高资源利用率,提高运算的并行性,减少资源等待时间,即可降低等待能耗,提高能源效率.

在分布式文件系统中,数据布局(data placement)是指数据在节点间按特定目标的分布状态[14],这一目标称为数据布局目标,实现该布局的算法称为数据布局算法.我们认为,一个良好的数据布局可以降低 MapReduce 系统的能耗.在 MapReduce 以及基于其的分布式文件系统(如 Hadoop HDFS)中,数据布局有 3 个特点:① 数据是分布的;② 数据在节点之间复制;③ 各个节点中的数据被并行地处理.数据布局决定了节点的数据特征(如数据量、值域分布等).在 MapReduce 系统中,一个作业会分为多个阶段,阶段内是并行执行的,阶段间是串行执行的,最为典型的就是 Map 阶段和 Reduce 阶段.同一阶段会包含多个任务(也有些文献把阶段称为任务,把任务称为实例),这些任务对数据的处理逻辑是一样的,它们在集群的各个节点上并行执行,且它们处理的数据集的大小、数据特征以及访问效率不同,因此结束时间也会不同.由于“迁移计算而非迁移数据”,计算被移动到距离数据最近的节点上运行,不同节点执行的任务算法相同,因此,任务的执行特征取决于输入的数据量以及数据特征,也即数据布局决定 MapReduce 任务(实例)的分发方式和执行效率.由此可见:要想提高任务的执行效率和并行性,减少节点间的等待,就必须尽可能地实现“负载均衡”的数据布局.本研究就是要定义这种数据布局目标,并找到实现这种数据布局的算法.

本文研究 MapReduce 系统中的数据布局特征,提出一种能耗优化的数据布局目标及实现算法,旨在降低 MapReduce 应用系统的能耗.数据布局是一个经典问题,在不同存储环境中有不同的设计目标,如公平性、自适应性、冗余性、容错性、时间有效性和空间有效性等.仅在 MapReduce 这种编程模型下,数据布局才会反过来决定计算特征,以能耗优化为目标的数据布局研究是一个崭新的研究问题.我们首先定义数据布局目标,即,一个能耗优化的数据布局应该满足的条件;随后描述在异构 MapReduce 集群环境下实现这一数据布局的具体算法;最后,我们通过理论和实验证明该算法实现的数据布局能够满足定义的目标,能够提高资源利用率,降低能耗.

本文第 1 节介绍相关工作.第 2 节和第 3 节分别阐述以能耗优化为目的的数据布局模型和数据布局算法.第 4 节从理论上证明数据布局算法能够实现既定目标.第 5 节对比本文提出的数据布局算法与广泛应用的一致性 Hash 算法的布局效果,并通过实验验证本文提出的数据布局对能耗的优化效果.第 6 节总结全文并提出下一步工作.

1 相关工作

首先,能耗优化并不是一个崭新的课题,国内外绿色计算或是高能效计算的研究成果很多,但从软件层面上

对计算机系统的能耗进行优化是近年来兴起的研究领域.以国内研究为例,近年来,部分重要的研究成果有文献[4−7,15−18],其中,MapReduce 系统能耗的研究有文献[4−7].国外的绿色计算研究开始较早,有许多成熟的优化技术,但 MapReduce 系统的能耗研究成果同样有限,以我们现有的知识,尚未发现通过数据布局来优化能耗的研究,与此最为相关的是通过数据迁移来优化能耗[13,15].文献[15]对数据可用性进行了建模,研究节点状态与数据块可用性之间的关系,提出构造数据块可用性度量矩阵,解决数据可用性完全覆盖问题,在不影响数据可用性的前提下,按负载规律使部分空闲节点休眠.雅虎开发的 GreenHDFS[13]系统能够根据文件的生命周期将文件从激活状态的节点移动到休眠的节点,以达到节能的目的.但它们[13,15]都属于动态的数据迁移算法而不是静态的数据布局算法,算法的有效性需要数据访问规律的支持,而我们提出的能耗优化方法并不需要依赖数据访问规律.

其次,其他分布式系统能耗优化方法可分为如下 3 类:

(1) 可通过调整 CPU 电压、改变 CPU 工作频率或嵌入式系统的工作状态达到节能的目的[19,20].此类方法与本研究弱相关.

(2) 部分通过动态作业调度使负载集中,关闭或休眠空闲的物理资源以达到节能的目的.例如:文献[21]设计了 Green-Net 框架管理大规模分布式系统的能耗,使用调度策略来关闭/休眠空闲设备,达到节能的目的;文献[17]利用排队模型对云计算系统进行建模,分析云计算系统的平均响应时间和平均功率,建立云计算系统的能耗模型,然后提出大服务强度和小执行能耗的任务调度策略,并使用调度策略降低云计算系统的能耗.

(3) 少量研究通过提高计算机资源利用率来降低能耗.例如,文献[22]根据 CPU 性能要求及 VMS 的当前资源状态和网络拓扑结构,通过虚拟机的资源分配和动态迁移机制提高能源利用率.

但这些传统方法[17,19−22]并不适用于 MapReduce 系统,原因如下:① 调整 CPU 状态会影响节点的计算性能,同时也降低了设备的寿命;② MapReduce 系统中各节点不仅是计算节点,而且是数据节点,关闭/休眠节点将使存储在该节点的数据不可用;③ 分配虚拟机资源的实施难度较大,并且在迁移虚拟机节点的同时也迁移了该节点上的数据,代价较大.我们提出的能耗优化方法通过静态的数据布局降低空闲能耗,数据布局算法并非反复执行,而数据布局对能耗的优化效果则是持久的.

在数据布局算法方面,国内外研究有:

(1) 优化大规模网络存储系统的并行度、容错性、可靠性

最具代表性的数据布局算法为一致性 Hash 算法[14].该算法将每个设备虚拟为 $k\log|N|$个设备,其中,k 为常数,N 为设备空间的设备总数.面对异构的集群,还需要将每个设备按照其比例虚拟成 $w_i/w_{\min}$ 个设备,其中,w_i 是第 i 个设备的权重,$w_{\min}=\min(w_1,w_2,\ldots,w_n)$.虽然使用该算法的数据布局使得网络存储系统具有极佳的动态自适应性和公平性,但添加的虚拟设备太多[23],对于存储系统来说难以承受,更不适用于大数据存储系统.文献[24]为了解决异构分布式系统中的数据布局问题,按照设备的剩余容量来分层,在每一层使用同构的数据布局算法,公平地将数据分布到设备上.文献[23]使数据布局算法适用于大规模网络存储系统,提出了一种能够公平、有效的数据布局算法 CCHDP.该算法将聚类算法和一致性 Hash 算法相结合,引入少量的虚拟设备,极大地减少了虚拟设备的个数.文献[25]提出了一种可靠的副本数据布局算法 RRDP,并在此基础上设计了一种面向多副本的自适应数据布局算法 RSEDP,然后提出了一种高效的分层数据布局算法 EHDP.此类数据布局算法设计之初并非针对 MapReduce 系统,设计目的并非能耗优化,这些算法可以运用到 MapReduce 系统中,同样可以提高并行性.但由于这些算法本身较为复杂,会在数据装载时引入额外的负载,因此能耗优化效果未必明显.本研究参照了这些算法的模型和设计思路,为降低算法本身带来的能耗开销,我们在保证布局公平性的前提下尽量简化布局算法.

(2) 降低数据密集型应用的网络传输

例如,文献[26]针对“云计算环境下面向流程的数据密集型应用中遇到的如何减少跨数据中心数据传输和保持数据间依赖性以及兼顾全局负载均衡”这一问题,提出了一种优化的数据布局算法.

(3) 对 RAID 磁盘阵列的改进

例如,Cortes 在文献[27,28]中分别对异构磁盘阵列的 RAID0 和 RAID5 结构的数据布局问题进行了研究;

文献[29]又对 Cortes 布局方法进行改进,提出了一种随机的块级存储虚拟化的异构存储系统,并可以让数据保持公平和冗余的特点.

后两类数据布局算法分别针对网络传输和磁盘存储,本文提出的布局算法与之有较大的区别.在MapReduce环境中,只要是公平的数据布局,都有利与提高并行性,进而优化能耗;本文提出的数据布局算法相对于现有算法来说,公平性会更好,而且算法简单,数据布局的代价小,更适合 MapReduce 系统的特性.

提高任务并行性,是本文优化 MapReduce 系统的一个重要途径.现有 MapReduce 的实现系统,如 Hadoop MapReduce,同样采用了一些任务调度相关的方法来保证并行性,但其能耗优化效果不佳.MapReduce 系统的任务副本机制在实现容错的同时提高了并行性,当一个任务执行完毕后,其副本的执行结果会被忽略.此外,MapReduce 还采用细粒度的任务来保证并行性,该方法简单且实际效果较好.对于同一任务,并行性最差的情况是所有节点等待最慢的节点执行最后一个任务,当任务很小时,这种等待也会随之减小.但是,上述方法均增加了额外的计算.副本机制实际上是增加了运算负载,而对于细粒度的任务,实际上是增加了调度和网络通讯的负载,这些都需要消耗额外的能量.以 Map 阶段为例,整个阶段执行时间取决于最后一个任务的结束时间,若采用细化 Map 任务大小的方法来保证 Map 任务的并行性,如默认 Map 任务的大小为“处理 64MB 数据”,则在最坏的情况下,其他 Map 节点则会等待“性能最差的节点处理 64MB 数据”所消耗的时间,这种等待是可以接受的.Map 任务越小,其并行性越好,但细分 Map 任务带来的调度开销不容忽视.假设 Map 阶段处理 1TB 的数据,则会有 16 384 个 Map 任务,每次任务分发的时间不大于 3s(默认心跳间隔为 3s),我们设为 1s,则表示 Map 节点将会有 1s 被动空闲时间来等待新任务,那么整个任务调度而产生的节点空闲时间为 16 400s,假设节点的空转功率为 60W,则会浪费 984KJ 的能量.而本文提出的从数据布局角度确保并行性,则可以避免细粒度任务带来的调度开销.

综上所述,在 MapReduce 系统能耗优化的研究中,还没有看到从数据布局角度设计的优化方法;而传统的数据布局算法并非针对能耗优化这一目标,也不能完全适用于 MapReduce 系统.因此,本文针对 MapReduce 系统特性设计能耗优化的数据布局算法,与现有工作存在区别.

2 数据布局模型

MapReduce 是迁移计算而不是迁移数据,它将计算迁移到数据所在节点而不是将数据迁移到计算节点.MapReduce 的输入数据一般来源于分布式文件系统(distributed file system,简称 DFS).DFS 上不同的数据布局将导致节点处理数据的代价不同,即,数据布局决定 MapReduce 处理数据时的能耗.本节将根据数据布局与能耗的关系,提出能耗优化目标和能耗优化的数据布局目标.

2.1 能耗优化目标

目前,能耗优化主要有两种思路:一是开发更加节能的硬件设备;二是采用软件方法,如动态迁移数据资源的位置、改变 CPU 的状态、动态调度作业以关闭或休眠节点等.我们则结合 MapReduce 系统集群的特点,提出能耗优化的新思路——减少“等待能耗”.

定义 1(MapReduce 系统能耗). MapReduce 系统的硬件层由相当数量的计算机节点组成,节点存储海量数据并执行数据分析作业,平台软件层面主要由 MapReduce 框架(如 Hadoop MapReduce)和分布式文件系统 DFS (如 Hadoop distributed file system)两个部分组成,应用层则由具体的数据分析应用(作业)组成.MapReduce 系统能耗是某应用(特指 MapReduce 作业)分析处理给定数据量(单位:MB)的能源消耗(单位:J).MapReduce 系统能耗受作业类型和数据量的约束,但本文研究具有普适性的能耗优化方法.

定义 2(等待能耗). 在 MapReduce 系统中,计算机节点(或计算机某组件)因等待其他资源而处于“被动空闲”的时间段内消耗的能量称为等待能耗.

例如,当分布式系统中某个节点在等待其他节点的运算结果时,产生等待能耗.该节点并非真正意义上的空闲,通常无法关闭该节点以节能,因此,我们应该尽量减少节点对资源等的等待,以减少等待能耗.同理,对于一个计算机内部的各个组成部分,如 CPU 运算,也会因等待 I/O 操作而阻塞,产生等待能耗.

等待能耗的定义和分析符合 MapReduce 的特点:首先,MapReduce 编程模型把作业分解成任务(如 Map 任务和 Reduce 任务)并部署到每个节点并行的执行,由于数据分布的特点和任务执行的特点,很可能会因任务结束时间不同步而造成部分节点等待(如 Reduce 任务需要等待所有 Map 任务执行结束);其次,MapReduce 执行的大部分是海量数据分析型作业,对于每个节点,CPU 的执行速度要远大于 I/O 读写速度,CPU 易因等待 I/O 操作而产生等待能耗,这一点在节点处理远程数据时尤为明显,网络数据传输速度较慢,导致其他组件大量空闲,产生等待能耗.

在 MapReduce 系统的平台层,DFS 中的数据分布式地存储在各个节点上,每个节点上的 Map 或 Reduce 任务优先处理本地数据,随后处理远程数据.处理远程数据的效率会明显低于处理本地数据的效率,导致 CPU 资源利用率不高,等待能耗增加.此外,无论数据来自网络还是本地,均会出现数据处理快的节点等待数据处理慢的节点,产生等待能耗.可以认为,MapReduce 与 DFS 不能实现良好的配合,导致额外的运算和能耗.

无论 Map 任务还是 Reduce 任务,其执行时间都取决于两个方面:一是计算复杂度,二是计算规模.但其能耗还取决于节点资源的利用率.数据布局决定了任务处理的数据量大小以及数据是来自本地节点还是远程节点.若数据按节点计算能力和数据特征公平地分布到各个节点,确保每个节点均处理本地数据,且同步完成本地数据的处理,则可以提高 CPU 的资源使用率和节点的并行性,降低等待能耗.综上,本文提出如下目标以达到能耗优化的目的:① 减少或消除任务处理远程节点数据,减少网络数据传输,提高 CPU 资源利用率;② 根据节点的计算能力,为每个节点公平地分配数据,使任务的执行时间相等,提高节点间的并行度.完成以上两个能耗优化目标,将有效地降低 MapReduce 集群的等待能耗,降低能耗.

2.2 异构MapReduce系统

本文描述的是数据布局的应用环境 MapReduce 系统,并且系统的硬件层由异构的计算机节点组成.本节将定义 MapReduce 系统的硬件环境及其组成.

定义 3(异构 MapReduce 节点). 一个 MapReduce 系统的硬件层 C 由众多异构的计算机节点 c_i 构成,即,$C=\{c_1,c_2,\ldots,c_n\}$,其中,n 为节点个数,下标表示节点按任意序从 1 至 n 线性编码.

定义 4(节点计算能力). 节点计算能力表征节点处理数据的能力,计算能力不仅与 CPU 运算速度有关,还与 I/O 能力相关.后者在大数据分析中更加重要.节点计算能力用正数 ω_i 表示,综合地考虑计算机的 CPU、内存、磁盘和网络设备,按公式(1)计算得:

$$\omega_i=g(c_i)=\phi_c(c_i)\oplus_1\phi_d(c_i)\oplus_2\phi_m(c_i)\oplus_3\phi_n(c_i) \tag{1}$$

对于每个节点 c_i,有且仅有一个 $\omega_i=g(c_i)$ 表示该节点的计算能力,g 函数为节点计算能力映射函数.本文研究异构的集群环境,即,$\exists c_i,c_j\in C, g(c_i)\neq g(c_j)$.$g$ 函数中,$\phi_c,\phi_d,\phi_m,\phi_n$ 分别是 CPU、磁盘能力、内存能力和网络设备能力的评价标准,$\oplus$是聚集运算符.由于各种设备的性能不一样,$\phi_c(c_i),\phi_d(c_i),\phi_m(c_i)$和$\phi_d(c_i)$的数值在量级上会有差距.因此,为保持各个设备的公平性,本文使用不同的运算符来调和它们之间的数值差距(见表 1).

Table 1 Aggregation operator for computing abilities of nodes

表 1 计算节点计算能力的聚集运算符

	使用方式	等价运算规则
$\oplus_1$	$Left\oplus_1 Right$	$Left+k_1\times Right$
$\oplus_2$	$Left\oplus_2 Right$	$Left+k_2\times Right$
$\oplus_3$	$Left\oplus_3 Right$	$Left+k_3\times Right$

我们认为,在数据处理的过程中,CPU 计算能力与磁盘读写能力同样重要,因此,

$$k_1=\frac{\max(\phi_c(c_1),\phi_c(c_2),\ldots,\phi_c(c_n))}{\max(\phi_d(c_1),\phi_d(c_2),\ldots,\phi_d(c_n))} \tag{2}$$

同样地,我们认为,内存是计算机多级存储的核心,是 CPU 与其他设备读写数据的媒介,它与 CPU 同等重要.因此,

$$k_2=\frac{\max(\phi_c(c_1),\phi_c(c_2),...,\phi_c(c_n))}{\max(\phi_m(c_1),\phi_m(c_2),...,\phi_m(c_n))} \tag{3}$$

但本文的优化方法将尽量减少甚至消除网络数据的传输,所以在数据处理时,MapReduce 系统基本不使用网络设备.因此,

$$k_3=\frac{\max(\phi_c(c_1),\phi_c(c_2),...,\phi_c(c_n))}{\max(\phi_4(c_1),\phi_4(c_2),...,\phi_4(c_n))}\times\varepsilon \tag{4}$$

公式(4)中,ε为一个小于 1 的数,ε的作用是弱化网络对节点计算能力的影响.因此,ε越小,数据布局的公平性越好.由后文的实验可知,ε会影响装载性能,当按数据布局算法为 MapReduce 系统装载数据时,节点装载性能取决于节点的网络传输能力,因此,ε越大,网络对节点计算能力的影响就越大,计算能力大的节点将会分配较多的数据,数据装载公平性更好.在本文的实验中,ε取 0.01.

$\phi_c(c_i)$,$\phi_d(c_i)$,$\phi_m(c_i)$和$\phi_n(c_i)$是设备计算能力的度量,由于⊕的存在,不同设备度量的量纲可以不同.本文分别使用公式(5)~公式(8)进行衡量:

$$\phi_c(c_i)=2\times C_f(c_i)\times C_c(c_i)\times C_{fn}(c_i)\times C_{mw}(c_i) \tag{5}$$

其中,$C_f(c_i)$表示节点 c_i 的 CPU 的频率,$C_c(c_i)$表示节点 c_i 的 CPU 的核心数量,$C_{fn}(c_i)$表示每周期浮点运算次数,$C_{mw}(c_i)$表示机器字长.详细介绍参见我们的前期工作[5].

$$\phi_d(c_i)=0.5\times(D_r(c_i)+D_w(c_i)) \tag{6}$$

其中,$D_r(c_i)$表示节点 c_i 的磁盘读速度,$D_w(c_i)$表示节点 c_i 的磁盘写能力.

$$\phi_m(c_i)=S(c_i) \tag{7}$$

其中,$S(c_i)$表示节点 c_i 的内存大小.

$$\phi_n(c_i)=0.5\times(N_r(c_i)+N_w(c_i)) \tag{8}$$

其中,$N_r(c_i)$表示节点 c_i 的网络读的速度,$N_w(c_i)$代表节点 c_i 的网络写的速度.

2.3 能耗优化的数据布局目标

本节将在异构 MapReduce 模型的基础上介绍能耗优化的数据布局目标.我们将一个基于 MapReduce 的大数据分析作业抽象为两个关键步骤:一是数据查询,二是在查询结果上的数据运算.在查询步骤中,节点对输入的数据进行遍历,并过滤掉未被选中的数据;在运算步骤中,节点对查询结果数据集(过滤后的数据)进行处理.

针对能耗优化目标,MapReduce 系统的最佳数据布局是每个节点拥有与自身计算能力相适应的数据量,各个节点只需处理本地数据,且处理时间相同.这种数据布局既能避免输入远程数据,避免节点等待网络数据传输,也能让各个节点的任务执行时间相同,避免节点间等待.上述目标均可以提高资源利用率,缩减等待能耗,降低能耗.考虑到任务包含对查询结果数据集的运算,因此,每个节点上的查询结果数据集大小也应该适应该节点的计算能力,这样,各个节点对结果数据集的运算时间也相等,进而保证任务执行时间相等.因此,最佳数据布局还需要使一个随机查询的结果数据集也公平地分布在各个节点之上.本节定义数据布局目标,首先给出如下定义:

定义 5(数据集与数据块). 将数据集 B 等数据量地随机分成 m 份,每一份是一个数据块 b_i,则数据集可以表示为 $B=\{b_0,b_1,\ldots,b_{m-1}\}$,其中,$m$ 为数据块的个数,下标表示节点按任意序从 0 至 m–1 线性编码.

定义 6(数据量公平性). 若该 MapReduce 系统的数据布局符合$\forall c_i\in C$,

$$\frac{m_i}{\sum_{k=1}^{n}m_k}=\frac{\omega_i}{\sum_{k=1}^{n}\omega_k} \tag{9}$$

则该 MapReduce 系统的数据布局符合数据量公平性.其中,$\omega_i=g(c_i)$,m_i 为数据块个数.

定义 7(数据值域公平性). 若该 MapReduce 系统的数据布局符合$\forall c_i\in C$:

$$\frac{h(c_i)}{\sum_{k=1}^{n}h(c_k)}=\frac{\omega_i}{\sum_{k=1}^{n}\omega_k} \tag{10}$$

则该 MapReduce 系统的数据布局符合数据值域公平性.其中,$h(c_i)$为随机数据查询在节点 c_i 上命中的数据块个数.

若数据布局满足数据量公平性,则各个节点上任务的数据查询阶段耗时相等;若数据布局满足数据值域公平性,则各个节点任务的数据运算阶段耗时相等.因此,任务只需读取本地数据,避免数据网络传输和节点间的等待,该数据布局就达到了能耗优化的目标.综上所述,能耗优化的数据布局目标为:MapReduce 系统的数据布局满足数据量公平性和数据值域公平性.

3 数据布局算法

本节提出能耗优化的数据布局算法,通过该算法可以使 MapReduce 系统的数据布局满足前文定义的数据布局目标.首先,本文给出数据布局算法的定义.

定义 8(数据布局算法). 数据布局算法 f 将 m 个数据块映射到异构 MapReduce 系统的 n 个节点上:

$$\forall b_i\in B,\exists! c_j\in C, c_j=f(b_i)(m>>n).$$

本文提出的能耗优化的数据布局算法记为 f_0.f_0 把按任意顺序线性编码的数据块集合$\{b_0,b_1,\ldots,b_{m-1}\}$分配到以任意顺序线性编码的节点集合$\{c_1,c_2,\ldots,c_n\}$,其中,分配规则为:$\forall b_i\in B,\exists! c_j\in C$,若 i 和 j 满足公式(11),则数据块 b_i 被分配到节点 c_j 上:

$$\begin{cases}\text{given } b_i \text{ and } c_j \ i\%\sum_{k=1}^{j}w_k-o_j\in[0,w_j] & ①\\ \text{and } \forall r\in(j,n]\ i\%\sum_{k=1}^{r}w_k-o_r\notin[0,w_r] & ②\end{cases} \tag{11}$$

其中,w_j 是节点的权重,根据节点的处理能力,按公式(12)计算得到:

$$w_j=\frac{\omega_j}{\min(\omega_1,\omega_2,...,\omega_n)} \tag{12}$$

在公式(11)中,使用 w 而不是ω的原因是为了保证有效的取模运算.ω的数值过大,数据块的个数 m 远小于ω,会导致公式(10)中 i 的取模运算结果总为 i.因此,我们将把ω按公式(12)等比例缩小为 w.由于 w 是正小数,因此,公式(11)中“%”为正小数的取模运算,且取模结果也为正小数.我们定义其运算方式为

$$x\%y=x-y\lfloor x/y\rfloor \tag{13}$$

公式(13)中,x 和 y 均为正小数,$\lfloor\rfloor$为向下取整.

公式(11)中,o_j 是节点随机偏移量,根据节点的权重计算得到:

$$o_j=\begin{cases}p_j\times\sum_{k=1}^{j-1}w_k, & j>1\\ 0, & j=1\end{cases} \tag{14}$$

$p_j\in[0,1)$为节点 c_j 的偏移比.每一个节点的偏移比是一个常量,仅初始化 1 次,且随机生成.由于取模运算会使数据布局算法丧失随机性,出现如第 1 个节点的数据会非常少等不公平情况.而随机性是数据量公平性和值域公平性的保证(具体证明见第 4.2 节).因此,公式(14)引入随机取值的偏移量.

f_0 算法步骤如下:

步骤 1. 对所有数据块以任意顺序线性编码为$\{b_0,b_1,\ldots,b_{m-1}\}$.

步骤 2. 对所有节点以任意顺序线性编码为$\{c_1,c_2,\ldots,c_n\}$.

步骤 3 正序遍历数据块$\{b_0,b_1,\ldots,b_{m-1}\}$且倒序遍历节点$\{c_1,c_2,\ldots,c_n\}$,对每一个数据块 b_i 和节点 c_j 的组合,执行步骤 4.

步骤 4. 若 b_i 与 c_j 符合公式(11)中的判定①,则执行步骤 5.

步骤 5. 分配 b_i 数据块至 c_j 节点.

倒序遍历保证公式(11)中的判定①是公式(11)的充分条件,因此只要判断公式(11)的判定①即可,提高算法效率.若采用正序遍历,则需判断公式(11)中的判定①和判定②,算法效率较低.

f_0 算法的伪代码表示如下:

算法 1. 数据布局算法 f_0.

Input:B,C.

Output:将 B 中的元素 b_i 分配到 C 中的元素 c_i 上.

Function f_0

1. **FOR** i=0 **TO** m−1
2. **FOR** j=n **TO** 1
3. **IF** $i\%\sum_{k=1}^{j} w_k - o_j \in [0, w_j]$ //公式(11)的判定①
4. distribute b_i to c_j
5. **CONTINUE**
6. **END IF**
7. **END FOR**
8. **END FOR**

图 1 展示了一个按照 f_0 进行的数据布局例子.该例将 9 个数据块$\{b_0,b_1,\ldots,b_8\}$分配到 3 个节点$\{c_1,c_2,c_3\}$上.数据块从 0 到 8 编号,节点从 1 到 3 编号.其中,1 号节点的权重 w_1=1,偏移比 p_1=0;2 号节点的权重 w_2=3,偏移比 p_2=0.7;3 号节点的权重 w_3=2,偏移比 p_3=0.5.

- 对于 0 号数据块:(0%6−0.5)∉[0,2],所以 $f_0(0)\neq 3$;(0%4−0.7)∉[0,3],所以 $f_0(0)\neq 2$;0%1∈[0,1],所以 $f_0(0)=1$;
- 对于 1 号数据块:(1%6−0.5)∉[0,2],所以 $f_0(1)\neq 3$;(1%4−0.7)∈[0,3],所以 $f_0(1)=2$;
- 对于 2 号数据块:(2%6−0.5)∈[0,2],所以 $f_0(2)=3$;
- 对于 3 号数据块:(3%6−0.5)∈[0,2],所以 $f_0(3)=3$.

依此类推,可以分配其他数据块到节点$\{c_1,c_2,c_3\}$上.计算能力最强的 2 号节点分得 4 块数据,次强的 3 号节点分得 3 块数据,最弱的 1 号节点分得 2 块数据,近似公平.本例仅展示算法的分配原理,当数据块个数远远大于节点个数时,算法公平性得以保证.

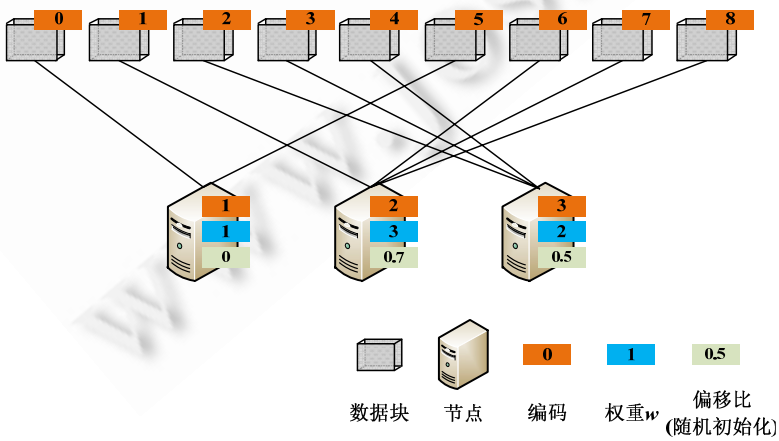


Fig.1 An example of energy consumption optimizing data placement algorithm

图 1 能耗优化的数据布局算法示例

数据布局算法在数据装载时执行,我们在后文的实验中证明了数据布局算法对数据装载能耗和性能的影响可忽略.此外,新增数据不会影响数据布局的公平性.在大数据环境中,更多的是数据的增加和访问.设现有数据集 D 已按本文提出的算法公平地在 MapReduce 系统中布局,若在此基础上新增数据集 d,且 d 同样采用本文提出的算法布局,那么最终的数据集 $D+d$ 在 MapReduce 系统中的布局也是公平的.由此可见,只要按照我们提出的算法装载新数据,在数据快速增长的环境中就仍然能够保持数据布局的公平性.

4 理论证明

本节将用理论方法证明能耗优化的数据布局算法 f_0 实现了数据布局目标,即,使用 f_0 布局数据后的 MapReduce 系统,其数据布局符合数据量公平性和数据值域公平性.证明思路如下:首先定义两种数据布局算法的等价性,随后提出一种与 f_0 等价的数据布局算法 f_0',并证明 f_0' 实现了数据量公平性和数据值域公平性,进而证明 f_0 同样实现该目标.

对于同一个 MapReduce 系统,判断数据布局算法 f_0 和 f_0' 是否等价,只需判断二者生成的数据布局是否一致,故有如下定义:

定义 9(数据布局算法等价性). 给定 MapReduce 系统 C 和数据集 B,若 $\forall b_i \in B, \exists! c_j \in C, f_1(b_i)=f_2(b_i)=c_j$,则数据布局算法 f_1 与 f_2 等价.

下面给出 f_0' 的实施步骤:

步骤 1. 对所有数据块以任意顺序线性编码为 $\{b_0,b_1,\ldots,b_{m-1}\}$.

步骤 2. 对所有节点以任意顺序线性编码为 $\{c_1,c_2,\ldots,c_n\}$.

步骤 3. 正序遍历节点 $\{c_1,c_2,\ldots,c_n\}$ 且正序遍历数据块 $\{b_0,b_1,\ldots,b_{m-1}\}$,对每一个数据块 b_i 和节点 c_j 的组合,执行步骤 4.

步骤 4. 若 b_i 与 c_j 符合公式(11)中的判定①,则执行步骤 5.

步骤 5. 若 b_i 已经被分配到 $\{c_1,c_2,\ldots,c_{j-1}\}$,则先取消对 b_i 的分配.执行步骤 6.

步骤 6. 分配 b_i 数据块至 c_j 节点.

其算法的伪代码如算法 2 所示.

算法 2. 数据布局算法 f_0'.

Input:B,C.

Output:将 B 中的元素 b_i 分配到 C 中的元素 c_i 上.

Function f_0'

```
1.  FOR j=1 TO n
2.    FOR i=0 TO m−1
3.      IF  i%∑_{k=1}^{j} w_k − o_j ∈[0,w_j]   //公式(11)的判定①
4.        IF b_i has been distributed to {c_1,c_2,…,c_{j−1}}
5.          cancel distribution of b_i
6.        END IF
7.        distribute b_i to c_j
8.        CONTINUE
9.      END IF
10.   END FOR
11. END FOR
```

f_0 与 f_0' 是两种不同的数据布局算法;f_0 是面向数据块集合的,倒序遍历节点;而 f_0' 是面向节点集合的,正序遍历节点.第 4 节论述了 f_0 的算法效率要高于 f_0' 的算法效率,但两者是等价的,证明如下:

证明:

1. 等价性证明.若数据布局算法 f_0 与 f_0' 的对数据块和节点的线性编码一致,则 f_0 与 f_0' 等价.

因为 f_0 与 f_0' 的对数据块和节点的线性编码一致.

所以 f_0 的步骤 1~步骤 3 与 f_0' 的步骤 1~步骤 3 一致.

又因为 f_0 和 f_0' 的步骤 4 一致,对于任意 b_i 和 c_j 的组合均需满足公式(11)中的判定①,且 f_0 的步骤 5 与 f_0' 的步骤 6 一致.

所以对于 f_0,判定①是公式(11)的充分条件,因此,b_i 和 c_j 在满足公式(11)时进行分配.对于 f_0',需要证明其步骤 5 满足公式(10)的判定②,则 f_0 的步骤 4、步骤 5 和 f_0' 的步骤 4~步骤 6 一致.使用反证法证明如下:

假设 f_0' 中数据块 b_i 与节点 c_j 组合不满足公式(11)时的判定②,则 b_i 必然已经被分配到 $\{c_1,c_2,\ldots,c_{j-1}\}$ 中的一个节点,那么步骤 5 取消了该分配.与假设矛盾.因此, f_0' 的步骤 5 满足公式(11)的判定②.

综上,若 f_0 与 f_0' 的对数据块和节点的排序和编码一致,则 f_0 与 f_0' 等价.

数据布局算法 f_0 与 f_0' 等价,基于此,本节将通过证明 f_0' 实现了数据布局目标来证明 f_0 实现了数据布局目标.要证明 f_0' 实现数据布局目标,首先要证明 f_0' 对所有数据块进行了分配,然后证明 f_0' 实现的数据布局既满足数据量公平性又满足数据值域公平性.

2. 完全分配证明.数据布局算法 f_0' 将所有数据块 $b_0,b_1,\ldots,b_{m-1}$ 分配到节点 $\{c_1,c_2,\ldots,c_n\}$ 上.

因为当 j=1 时,o_1=0,步骤 4 中公式(11)的判定①必然被满足:

$$i\%\sum_{k=1}^{1}w_k-o_1=i\%w_1\in[0,w_1] \tag{15}$$

所以 f_0' 将所有的数据都落在编号为 1 的节点上.

又因为当 $j\neq1$ 时,根据步骤 5,只有符合条件的数据块重新分配到节点 j 上,而不会从 $\{c_1,c_2,\ldots,c_n\}$ 中取出数据块,所以 f_0' 将所有数据块 $\{b_0,b_1,\ldots,b_{m-1}\}$ 分配到系统 $\{c_1,c_2,\ldots,c_n\}$ 上.

综上, f_0' 完全分配.

3. 数据量公平性证明.数据布局算法 f_0' 实现数据量公平性.

f_0' 的分配过程可以看成是往系统中添加节点 c_j,然后将需要分配的数据块重新迁移到节点 c_j 的过程.下面使用数学归纳法来证明.

- 当 j=1 时,视为只有 1 个数据节点, f_0' 将所有的数据都分配到 c_1 上(见第 2 部分证明),实现数据量公平性.
- 当 j>1 时,设系统中有 j–1 个节点时, f_0' 实现数据量公平性.求证系统中有 j 个节点时, f_0' 实现数据量公平性.

图 2 展示了 f_0' 分配到 c_j 的数据块的规律.m 个数据块被划分为⌈m/len_j⌉段,除最后一段外,每段长度 len_j.被选中放入 c_j 的数据块编号从 o_j 到 o_j+w_j.首先,对于新添节点 c_j,分配到 c_j 的数据块的个数为 w_j/len,c_j 的数据量符合其计算能力;其次,由于偏移比 p_j 是随机初始化的常量,因此偏移量 o_j 是随机的,即,在每一个长度为⌈m/len⌉的段里面,被选取分配到 c_j 的数据块是随机的,不损害前 j–1 个节点的数据量公平性.

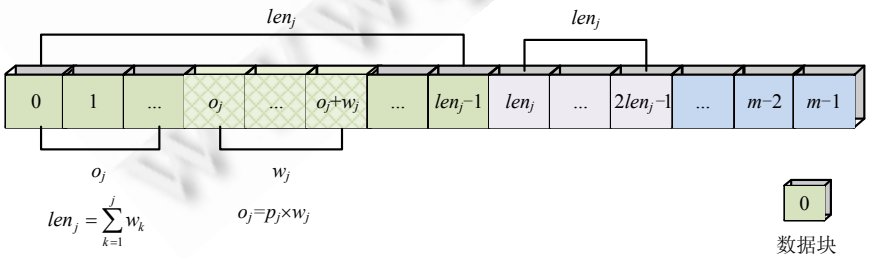


Fig.2 Data blocks which are distributed to c_j

图 2 分配到节点 c_j 的数据块

综上,数据布局算法 f_0' 实现数据量公平性.

4. 数据值域公平性证明.

数据布局算法 f_0' 实现数据值域公平性.

由定义 7 可知,数据值域公平性本质上是查询结果数据集的数据布局满足数据量公平性.若 f_0' 实现的数据布局具有随机性,则任意查询选中的数据块必然随机地分布在每个节点上.在这种情况下,若公式(9)成立,则公式(10)一定成立:

$$\frac{m_i}{\sum_{k=1}^{n} m_k}=\frac{h(c_i)}{\sum_{k=1}^{n} h(c_k)}=\frac{\omega_i}{\sum_{k=1}^{n} \omega_k} \tag{16}$$

因此,要证明 f_0' 实现数据值域公平性,只需证明 f_0' 分配数据块到节点的过程具有随机性.由第 2 部分的证明可知,当 j=1 时,视为只有 1 个数据节点,f_0' 将所有的数据都分配到 c_1;当 j>1 时,c_j 从$\{c_1,c_2,\ldots,c_{j-1}\}$中随机地获取数据块.因此,$f_0'$ 分配数据块到节点的过程具有随机性.

综上,数据布局算法 f_0' 实现数据值域公平性. □

由证明 2~证明 4 可知,f_0' 实现数据布局目标.又因 f_0 与 f_0' 等价,因此,f_0 实现数据布局目标.

5 实验验证

前一节对数据布局算法是否能满足能耗优化目标进行了理论证明,但这并不能验证该数据布局算法比经典算法更优,也不能证明数据布局能够优化 MapReduce 系统的能耗.为进一步验证数据布局算法的有效性,本节首先对比该算法和一致性 Hash 算法,并在异构 MapReduce 系统环境中对其能耗优化效果进行分析和比较.

5.1 算法对比

一致性 Hash 算法目前在分布式存储系统中应用广泛,特别是对于解决热点问题的 Cache 存储系统.扩展一致性 Hash 算法可用来实现数据布局目标,但面对处理大数据应用的 MapReduce 系统,一致性 Hash 算法对比本算法,缺点也十分明显.本文比较了数据布局算法和两种一致性 Hash 算法经典扩展的布局均衡性效果和性能:

- 一种是在异构节点上的一致性 Hash 算法扩展 h_1.

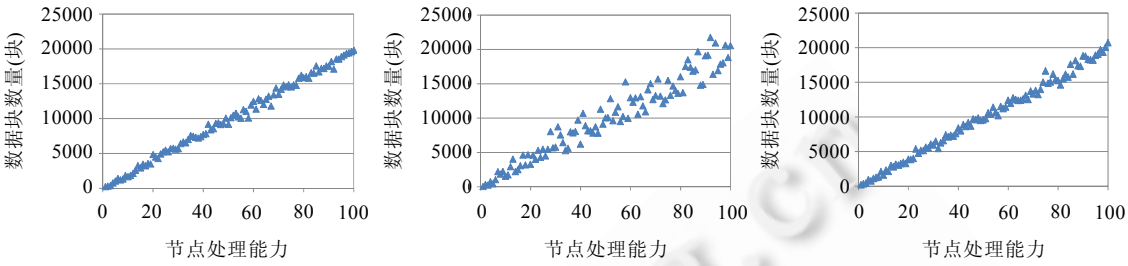
面对异构节点,需要将每个设备按照其比例虚拟成 $w_i/w_{\min}$ 个设备,其中,w_i 是第 i 个设备的权重,$w_{\min}=\min(w_1,w_2,\ldots,w_n)$,这就使得系统建立在 $\sum_{k=1}^{n}\frac{w_i}{w_{\min}}$ 个节点上,其数据量公平性取决于虚拟节点的个数,虚拟节点数越少,其数据量公平性越差.在节点处理能力差距较大的系统中,一致性 Hash 算法的空间复杂度和时间复杂度很大,而此类系统在复杂的云环境中则非常常见.

- 另一种是在 h_1 的基础上,为了防止节点个数过少而造成随机映射不均衡的扩展方式 h_2.

一致性 Hash 算法将每个节点再虚拟成 $k\log|N|$(k 为常数,N 为数据块个数)个节点,后文称该过程为数据公平性加强的一致性 Hash 算法和 h_2.因此,本节将对比本文提出的数据布局算法 f_0、未进行数据量公平性加强的一致性 Hash 算法 h_1 和进行数据量公平性加强的一致性 Hash 算法 h_2(k=1).

首先,本文提出的数据布局算法 f_0 的数据量公平性远好于未进行数据量公平性加强的一致性 Hash 算法 h_1,略好于进行数据量公平性加强(k=1)的一致性 Hash 算法 h_2.因为在将节点映射到环形区域的过程中,若节点个数较少,随机方法并不能保证相邻两个节点组成的区间大小大致相同,从而造成随机因素对 h_1 的数据量公平性影响较大.进行数据量公平性加强,即添加了大量虚拟节点以后的 h_2,随机因素对一致性 Hash 的数据量公平性影响变小.图 3 显示了 3 种算法对比的实验结果:横坐标表示节点的处理能力,纵坐标表示节点上分配的数据块个数.可以证明:节点处理能力和数据块数量成正比的线性关系越强,数据量公平性越好.从图 3 中可以看到,图 3(a)的公平性远好于图 3(b)、略好于图 3(c).为了定量地比较,我们使用 PEARSON 相关系数(取值范围为[–1,1])来衡量线性关系:两个变量之间的线性关系越强,PEARSON 相关系数的绝对值就越大.PEARSON 相关系数大于 0 表示正相关,小于 0 表示负相关.经统计可知,图 3(a)的 PEARSON 相关系数为 0.997 455 423,图 3(b)为 0.971 928 44,图 3(c)为 0.996 455 194.图 3 中,点的聚集方式的对比和 PEARSON 相关系数的对比都验证了 f_0 的数据量公平性

远好于 h_1,略好于 h_2.

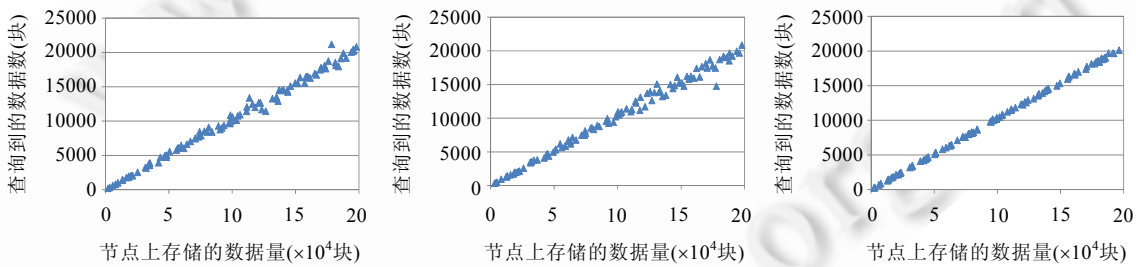


(a) 数据布局算法 f_0 (b) 一致性 Hash (未进行数据量公平性加强) (c) 一致性 Hash (进行数据量公平性加强,k=1)

Fig.3 Comparison of algorithms in data amount fairness

图 3 算法数据量公平性对比

其次,本文提出的数据布局算法 f_0 的数据值域公平性优于未添加大量虚拟节点的一致性 Hash 算法 h_1.h_1 和 f_0 同样具有随机性,为其在数据值域公平性上提供了必要的条件.但由证明过程中公式(16)可知,数据值域公平性的必要条件还有数据量的公平性.从随机性上说,f_0 与 h_1 存在差距,而 f_0 的数据量公平性远优于 h_1.图 4 显示了在同样的数据、同样的查询条件和同样的节点处理能力下,3 种数据布局算法的值域公平性相差无几.通过详细数据统计得到 f_0,h_1 和 h_2 的 PEARSON 相关系数分别为 0.996 390 657,0.995 222 71 和 0.999 872 255,h_1 的值域公平性略低于 f_0,h_2 的值域公平性略优于 f_0.



(a) 数据布局算法 f_0 (b) 一致性 Hash (未进行数据量公平性加强) (c) 一致性 Hash (进行数据量公平性加强,k=1)

Fig.4 Comparison of algorithms in data codomain fairness

图 4 算法值域公平性对比

再次,一致性 Hash 算法 h_1 和 h_2 有着很好的适应性,添加或删除节点都不会导致大量数据的迁移,但损失布局公平性.本文提出的数据布局算法 f_0 在增加节点的情况下,移动的数据量与 h_1 和 h_2 相同;删除节点需要移动的数据量高于 h_1 和 h_2.然而,本文提出的算法在适应节点数量发生的变化后,布局公平性不会发生变化;而每增加或删除一个节点时,h_1 和 h_2 仅简单地将邻近的数据移动到拟增加的节点,或将拟删除节点的数据移动到邻近的节点,在未添加大量虚拟节点的情况下,每增加或删除一个节点,其数据量公平性和值域公平性都将变差.

最后,虽然一致性 Hash 算法 h_2 可以通过添加大量的虚拟节点来弥补数据量公平性缺陷,但代价是增加算法时间和空间复杂度.系统不但需要存储更多的虚拟节点,而且需要对这些虚拟节点进行 Hash 运算和在这些虚拟节点中进行数据块的分配.对比实验中,f_0,h_1 和 h_2 这 3 种算法的运行时间分别是 45s,30s 和 165s.可见,即使通过添加大量虚拟节点,分配算法的执行时间超过本文提出的数据布局算法 3 倍,其均衡效果也未能优于本文提出的算法.

综上所述,尽管一致性 Hash 算法 h_1 在添加了大量虚拟节点以后,在数据量公平性和值域公平性上具有一定

的优势,但算法时间和空间上的损失难以忽视,这就导致一致性 Hash 本身并不适用于处理大数据的 MapReduce 系统.而本文提出的数据布局算法 f_0 恰恰只需要为每个节点存储少量数据且不添加虚拟节点,即可达到能耗优化的数据布局目标,其效果优于 h_1,等同与 h_2;同时,在执行效率上优于 h_2.从适应性角度支持系统规模的扩大,而当系统的规模缩小(较少发生)时,f_0 适应性有待提高.

5.2 实验环境

我们采用基于 Hadoop HDFS 和 MapReduce 的异构实验环境,系统由 6 个节点构成,包含 1 个 JobTracker 节点和 5 个 TaskTracker 节点.能耗测试实验主要针对执行任务的 TaskTracker,我们选用了两种计算能力差距较大的计算机作为 TaskTracker.表 2 描述了系统中节点配置.

Table 2 Heterogeneous environment for energy consumption experiments

表 2 异构的能耗实验环境

节点	类型	数量	描述	
JobTracker	类型 1	1	计算机	清华同方超翔 Z900 计算机
			CPU	Inter Core i5-2300 2.80GHz,100MHz 外频,28X 倍频
			内存	8GB
			硬盘	1TB
			网卡	千兆网卡
TaskTracker	类型 1	3	计算机	清华同方超翔 Z900 计算机
			CPU	Inter Core i5-2300 2.80GHz,100MHz 外频,28X 倍频
			内存	8GB
			硬盘	1TB
			网卡	千兆网卡
	类型 2	2	计算机	清华同方超翔 C3001 计算机
			CPU	Inter Core Pentium(R) 4 3.00GHz,200MHz 外频,15.0X 倍频
			内存	1GB
			硬盘	80GB
			网卡	百兆网卡

根据表 2 的实验环境,我们在 MapReduce 系统中对不同任务量的 WordCount(CPU 密集型计算)[30],*Sort*(I/O 密集型计算)[31]和 MRBench(交互式计算,小作业高并发地执行)[32]进行了测试,研究其能耗优化效果.*WordCount* 是一种 CPU 密集型运算,网络和磁盘 I/O 负载较轻;Sort 是一种 I/O 密集型运算;MRBench 是交互式运算,较小的作业(jobs)高并发地执行.事实上,3 种运算都是数据密集型运算,所谓 CPU 密集型计算、I/O 密集型计算和交互式计算均为相对密集型.为对比能耗优化的效果,系统部署了 3 种不同的 MapReduce 系统,并让它们分别运行上述测试用例,确保每个作业的输入数据、处理逻辑和输出数据都相同.最后对实验结果进行分析,以验证数据布局能够有效地降低能耗.

我们基于 Hadoop MapReduce 构建了两种 MapReduce 实现系统,分别称为 LocalHadoop 和 NeoHadoop.三者均基于 Hadoop 分布式文件系统,但 LocalHadoop 删除了原生 Hadoop MapReduce 的远程数据访问部分,任务仅处理本地数据;NeoHadoop 则使用本文提出的数据布局算法 f_0 对数据进行布局.我们对比 Hadoop, LocalHadoop 和 NeoHadoop 执行不同算法的能耗、磁盘读写、网络读写等信息,以观察本文提出的数据布局是否优化了 Hadoop 的能耗.Hadoop,LocalHadoop 和 NeoHadoop 的区别见表 3.

Table 3 Differences of three systems which implement MapReduce

表 3 MapReduce 的 3 种实现系统的区别

系统名称	MapReduce 模块	分布式文件系统模块	数据布局算法	是否具有调度器	数据来源
Hadoop	Hadoop MapReduce	Hadoop DFS	随机	是	本地/网络
LocalHadoop	修改的 Hadoop MapReduce	Hadoop DFS	随机	否	本地
NeoHadoop	修改的 Hadoop MapReduce	Hadoop DFS	f_0	否	本地

5.3 实验结果

我们首先比较了 Hadoop,LocalHadoop 和 NeoHadoop 装载数据的能耗;随后比较 3 种系统运行 WordCount, Sort 和 MRBench 应用的能耗,证明优化效果;最后,对于每一个应用,我们研究相同任务量下 3 个系统的累积磁盘 I/O、网络 I/O 和 CPU 资源使用量规律,阐明数据布局能够优化 MapReduce 系统能耗的原因.本节考察的测量值见表 4.

Table 4 Measurement approaces of experimental results

表 4 实验值的测量方法

参数	单位	范围	计算方法
能耗	Joule	系统平均值	北电电力监测仪(专业版),依据 GB/T17215-2003,功率误差值±0.01~0.1W,功率测量上限 2 200W,数据采样频率在 1.5s~3s 之间
磁盘 I/O	MB	给定任务的执行时间内,系统累积值	所有节点每秒测量的磁盘 I/O 值在任务执行时间上的累加
网络 I/O	MB		所有节点每秒测量的网络 I/O 值在任务执行时间上的累加,考虑数据在网络中中传递,因此总量除以 2
CPU 负载	GHz		所有节点每秒测量的 CPU 频率和使用率的乘积在任务执行时间上的累加

表 4 中的测量数据均采用了归一化,设 Hadoop 系统的测量值为单位 1,LocalHadoop 和 NeoHadoop 的测量值均为与 Hadoop 系统的测量值的比例.

实验 1. 3 种系统的装载能耗以及性能比较.

首先,数据装载仅需 1 次,因此数据布局算法也不会反复执行,其后的数据处理都将受益于公平性数据布局,平摊后的数据布局算法代价很小;其次,从性能角度来看,3 个系统的数据装载均以数据块为最小单位.判断某数据块是否分配到该节点的时间代价,要远小于将该数据块通过网络装载到某节点的时间代价.若采用两个线程完成数据布局和装载,其中,布局线程计算数据块与节点的对应关系,装载线程负责传输数据到对应的节点,那么传输线程将永远不会因等待计算线程而阻塞.因此理论上,装载速度取决于各节点之间的网络传输能力,数据布局算法的性能代价可以忽略.考虑本文提出的数据布局算法会在数据装载过程中执行额外的运算,因此,本实验评价 Hadoop,LocalHadoop 和 NeoHadoop 这 3 种系统数据装载相同数据量的系统能耗和装载时间.实验结果如图 5 所示.

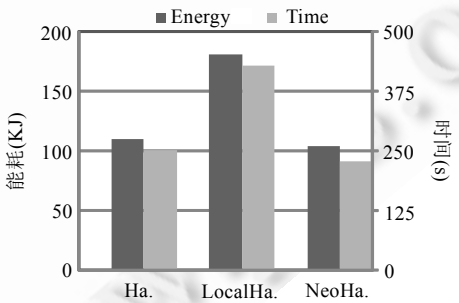


Fig.5 Comparison of average energy consumption and execution time of data loading of three systems

图 5 3 种系统的装载平均能耗和装载时间对比

从图 5 可以看出,Hadoop 和 NeoHadoop 系统的装载能耗和性能都相差无几.由于测量难免存在较小的误差和偶然性,因此可以认为,NeoHadoop 执行本文提出的数据布局算法并未影响数据装载能耗和性能.然而对于 LocalHadoop,由于没有采用均衡的数据布局算法,因此每个节点的数据量近似相同,而系统中性能较差且网络速度较慢的节点的装载速度较慢,导致较快的节点等待较慢的节点,因此产生大量等待时间和等待能耗,LocalHadoop 的装载能耗和性能低于 Hadoop 和 NeoHadoop.数据装载是一种简单的应用,在装载过程中计算机功率较稳定,装载能耗和装载性能主要取决于节点本身的数据吞吐量.由于数据是通过网络加载的,因此决定装载性能的主要因素是节点网络带宽.图 6 给出了装载相同数据时,3 种系统的资源使用情况.

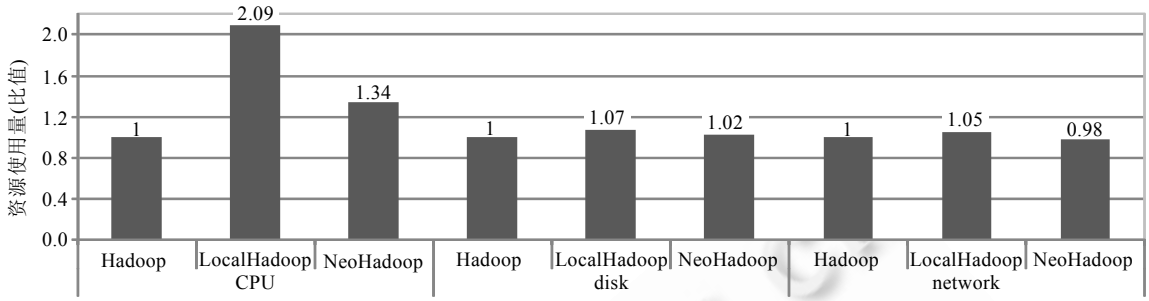


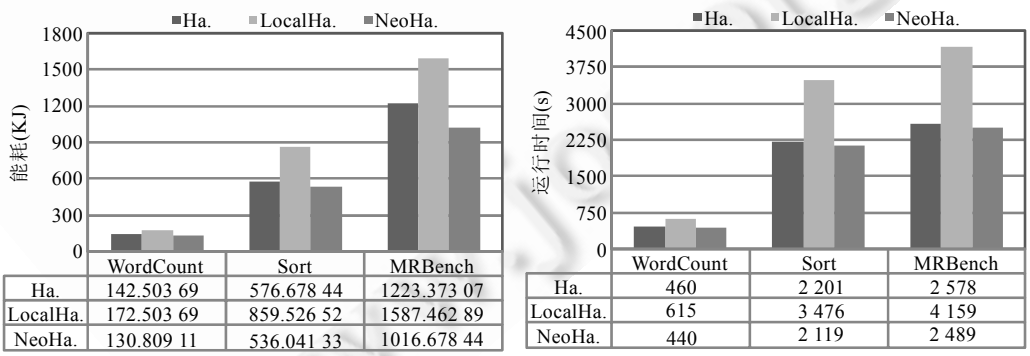
Fig.6 Comparison of accumulated resource usage (normalized) in data loading of three systems

图 6 3 种系统的装载时累积资源使用量对比

从图 6 可以明显得出,3 种系统的本地 I/O 和网络 I/O 资源使用量近似相等.这是由于装载数据集一致,且我们关闭了文件系统的副本机制,因此,I/O 资源使用量等同于装载的数据量.NeoHadoop 的 CPU 负载略高于 Hadoop,这是因为 NeoHadoop 系统需要额外的线程计算数据布局,因此消耗了部分 CPU 资源.而 LocalHadoop 的 CPU 负载高于其他两者的原因是:对于计算能力差的节点的 CPU 会因为等待网络数据传输而空闲,浪费 CPU 资源;计算能力强的节点会因等待计算能力差的节点完成数据装载而空闲,同样浪费 CPU 资源.此外,无论是磁盘 I/O 还是网络 I/O 资源,LocalHadoop 均略大于 Hadoop 和 NeoHadoop.这是因为 LocalHadoop 的装载执行时间长于其他两者,因此节点间的心跳检测等通信代价、Hadoop 文件系统日志以及实验采用的数据采集程序的本地日志均带来了额外的 I/O 开销.综上所述,理论分析以及图 5 和图 6 的实验数据表明,本文提出的数据布局开销很小,优化数据布局对数据装载能耗的影响可忽略.

实验 2. 3 种系统的累积能耗以及性能对比.

图 7 对比了 Hadoop,LocalHadoop 和 NeoHadoop 在 wordcount,sort 和 mrbench 这 3 种作业下,系统的平均能耗和性能.由于本文的研究对象是包含软件和硬件的 MapReduce 系统,如定义 1 所述,能耗是与算法相关的,因此我们仅能比较同一作业不同系统的能耗和性能以及不同作业对能耗优化效果的影响,但难以比较不同作业的系统能耗和性能.



	WordCount	Sort	MRBench
Ha.	142.503 69	576.678 44	1223.373 07
LocalHa.	172.503 69	859.526 52	1587.462 89
NeoHa.	130.809 11	536.041 33	1016.678 44

(a) 能耗

	WordCount	Sort	MRBench
Ha.	460	2 201	2 578
LocalHa.	615	3 476	4 159
NeoHa.	440	2 119	2 489

(b) 执行时间

Fig.7 Comparison of average energy consumption and execution time of WordCount, Sort and MRBench on three systems

图 7 3 种系统执行 WordCount,Sort 和 MRBench 的平均能耗和运行时间对比

- 首先,NeoHadoop 的能耗要优于 Hadoop 和 LocalHadoop.
  - 与 Hadoop 的能耗相比,在 WordCount 和 Sort 下,NeoHadoop 优化能耗约 8%;而对于 MRBench 作业,优化能耗达 16%.后文将证明,能耗降低的主要原因有:① 由于 NeoHadoop 中数据按数据

量和值域均衡布局,避免了任务处理远程数据,CPU 不会等待网络 I/O,减少了等待能耗;② 由于 NeoHadoop 中数据布局考虑了节点计算能力的差异,避免计算能力强的节点等待计算能力差的节点,减少了因节点间等待产生的等待能耗.

- 此外,尽管 LocalHadoop 系统不存在网络数据读取(Shuffle 阶段除外),理论上 CPU 不会因等待网络 I/O 而阻塞,产生等待能耗,但由于随机的数据布局策略导致每个节点处理的数据量近似相等,于是,LocalHadoop 在执行 MapReduce 作业时,性能较高的节点会因等待性能较差的节点而空闲,产生大量等待能耗.

- 而在性能上,NeoHadoop 也要优于 Hadoop 和 LocalHadoop,我们将在第 5.4 节分析能耗优化和性能优化之间的关系.
- 此外,从实验结果可知,LocalHadoop 的能耗明显要高于 Hadoop,节点间的并行性损失是能耗较高的主要原因;
- 最后,如图 7(a)所示,在 MRBench 下,NeoHadoop 的能效能耗优化效果要明显由于其他两个作业.这是因为 MRBench 由很多小 MapReduce 作业组成,对于 Hadoop 和 LocalHadoop,每个 MapReduce 作业均会产生网络 I/O 等待和节点间等待,而 NeoHadoop 的优化机制则多次避免这些等待发生,因此,能效能耗优化效果比单个作业更明显.

图 8 给出了 3 种系统执行 WordCount,Sort 和 MRBench 的累积资源使用量(CPU,磁盘 I/O,网络 I/O)对比.

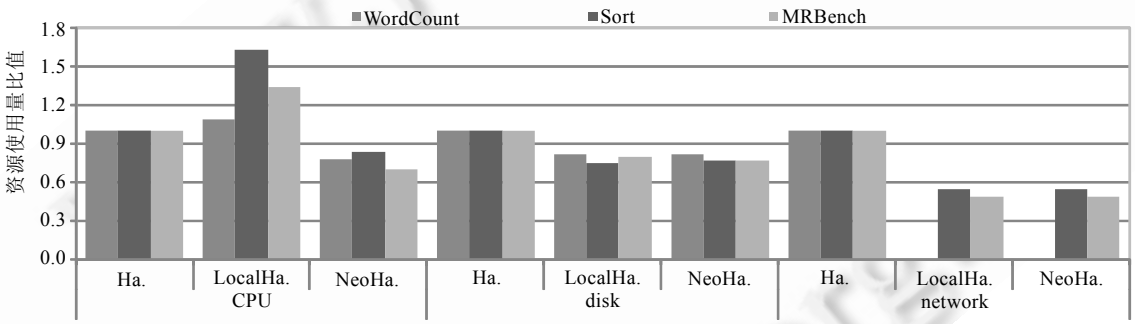


Fig.8 Comparison of accumulated resource usage (normalized) of WordCount, Sort and MRBench on three systems

图 8 3 种系统执行 WordCount,Sort 和 MRBench 的累积资源使用量(归一化)对比

由于资源使用量是与算法相关的,因此我们仅能比较同一作业在系统中的资源使用量,但难以比较不同作业的资源使用量.

- 首先,我们分析 CPU 资源使用量.

CPU 使用量可以表征等待能耗的大小,对于相同的算法和相同的数据量,CPU 资源使用量越大,表明 CPU 因空闲而空转的时间越长,等待能耗越大.图 8 中,WordCount,Sort 和 MRBench 的资源量都满足 NeoHadoop< Hadoop<LocalHadoop 的关系,这是因为,Hadoop 中并行性造成的 CPU 空闲较少,网络 I/O 造成的 CPU 空闲;LocalHadoop 中行性造成的 CPU 空闲较多,网络 I/O 造成的 CPU 空闲较少;而 NeoHadoop 中两者均很少;并且节点等待产生的等待能耗要大于网络 I/O 等待产生的等待能耗.此外,在 MRBench 下,NeoHadoop 的 CPU 资源使用量优化效果明显要优于其他两个作业,这与图 7 的分析结果一致.

- 其次,我们分析磁盘 I/O 资源使用量.

对于每一个节点,无论任务是处理本地数据还是网络数据,都会在某一节点上产生相应的本地数据读取,都消耗一定的磁盘 I/O,若处理网络数据,还要产生额外的网络 I/O.因此,磁盘 I/O 资源使用量即表征处理的总数据量(还包括少量的虚拟内存和中间数据存储).无论何种作业,3 种系统的磁盘 I/O 资源使用量都应该近似相等.对于节点等待时间较长的 LocalHadoop,日志程序会带来额外较少的本地 I/O 开销.图 8 的实验结果证实了这一点.

- 最后,我们分析网络 I/O 资源使用量.

对于 NeoHadoop 和 LocalHadoop,由于均保证了任务处理本地数据,因此网络 I/O 均来自于 MapReduce 的 Shuffle 阶段;对于 Hadoop,网络数据来自 Map 任务处理远程数据以及 Shuffle 阶段.由此可知,图 8 中对于每一个作业,NeoHadoop 和 LocalHadoop 的网络 I/O 资源使用量应大致相等并明显小于 Hadoop.由于 WordCount 中的 Shuffle 阶段仅需要合并很少的数据,因此在 WordCount 下,NeoHadoop 和 LocalHadoop 的网络 I/O 资源使用量非常小.

对图 7 和图 8 的实验结果分析可知:① LocalHadoop 只处理本地任务,且数据布局是随机的,在节点计算能力差距很大的异构 MapReduce 系统中,必然会产生计算能力强的节点等待计算能力差的节点,造成节点间等待;② Hadoop 通过远程数据处理,在一定程度上避免了节点间的等待,但网络数据读写代价是造成等待能耗的主要原因;③ 由于数据的公平性布局,NeoHadoop 的节点间并行性很好,并且不需要通过远程数据处理来保证并行性.NeoHadoop 利用公平的数据布局,其能耗低于 Hadoop 是因为避免了网络数据读写,而低于 LoaclHadoop 是由于避免了节点间等待.

5.4 能耗-性能分析

对于一个软、硬件整体考虑的系统,电能既可以认为是硬件直接消耗的,也可以认为是软件间接消耗的.由于能耗和性能之间的关系,能耗优化研究均伴随着性能的变化.现有研究对能耗优化和性能优化之间的关系持两种截然相反的观点:

- 在一些研究中,能耗优化和性能优化负相关,通常需要损失性能来换取低能耗,这符合硬件的设计思路:硬件工作在高性能状态时功率也高;反之,若降低性能则会获得较低的能耗.所以,它们之间存在着某种折中关系[33].
- 另一些文献则声称能耗优化和性能优化是一致的,通过资源分配和任务调度,使资源充分利用,减少因资源浪费而产生的额外能耗.这种优化方法使更多的有效资源投入到计算中,能够同时提高计算性能[34].

本文则主要采用后一种优化思路.

我们提出的能耗优化方法的核心思想是“减少各个节点之间和节点内部的等待所产生的不必要的等待”.本文采用能耗作为优化对象,因此将上述“等待”定义为“等待能耗”;若采用性能作为优化对象,则上述“等待”可以定义为“等待时间”.因为硬件设备在空等时的空载功率是稳定的,而能耗等于功率乘以时间,因此降低“等待能耗”也会同时降低“等待时间”,性能优化比例和能耗优化比例应该相同.然而,我们在优化等待能耗的同时减少了网络数据读写,因此减小了网络 I/O 设备(如网卡)的工作时间,降低其工作负载(功率),网络 I/O 设备的能耗降低,进而整体能耗优化比例会大于性能优化比例.由图 7 可知,对于 WordCount,Sort 和 MRBench,能耗优化比例分别为 8.2%,7%和 16%,性能优化比例分别为 4.3%,3.7%和 3.4%.能耗优化效果明显优于性能优化效果.

5.5 实验结论

总结前一节的实验分析,我们可以得出如下的实验结论:

(1) 本文提出的数据布局算法达到了能耗优化的数据布局目标,其效果远于未进行数据量公平性加强(通过添加大量虚拟节点)的一致性 Hash 算法的效果,略优于与进行数据量公平性加强的一致性 Hash 算法效果.然而一致性 Hash 算法添加大量虚拟节点,因此时间和空间复杂度显著增加,并不适用于 MapReduce 系统.

(2) 本文提出的数据布局能够提高 MapReduce 系统的能耗,优化效果明显,性能也随之优化.

(3) 在异构的环境中,Hadoop MapReduce 系统中,数据处理快的节点不断从慢的节点上获取数据,而传输过程中,节点的运算设备等待网络设备而产生等待能耗.

(4) 对比 Hadoop 和 LocalHadoop,LocalHadoop 降低了节点内设备间的等待能耗;对比 LocalHadoop 和 NeoHadoop,NeoHadoop 降低了节点间的等待能耗.

(5) 对比 Hadoop 和 LocalHadoop,节点计算能力差异越大,设备间的等待能耗越大,造成 LocalHadoop 的能耗高于 Hadoop 的能耗.

(6) NeoHadoop 只读取本地数据,降低了节点的设备间等待能耗.同时,NeoHadoop 的每个节点输入的数据量是公平的,使得 NeoHadoop 降低了节点间等待能耗.这两方面使得 NeoHadoop 能耗低于 Hadoop 和 LocalHadoop.

(7) 数据布局算法并没有给数据装载过程增加过多的性能开销.同时,可以通过独立启动数据布局线程来减小这种开销对能耗的影响.

6 结论和下一步工作

本文针对 MapReduce 系统的特点,首先提出了不同于传统能耗优化的新思路,在此基础上,提出了具体的能耗优化的目标,定义了 MapReduce 系统的等待能耗和优化方法;然后提出了能耗优化的数据布局目标以及实现该目标的数据布局算法;最后,从理论角度证明了数据布局算法的正确性,从实验角度验证了数据布局对 MapReduce 系统能耗优化的效果,并和经典算法进行比较,总结实验结论.本文提出的能耗优化方法适用于基于 MapReduce 的大数据分析应用程序,上述工作都将促进高能耗计算和大数据分析的应用,也将指导基于 MapReduce 系统的大数据分析系统的研发.本研究还处于初步阶段,在某些方面还存在不足.例如,MapReduce 系统中不可避免地会出现节点失效,数据布局算法并不能保证在节点失效时,迁移数据的代价最小,尽管数据副本机制可以部分地解决这一问题,但所提出的数据布局算法尚缺少适应性支持.我们将依据本文给出的优化思路,针对这些不足之处,做进一步的研究.

References:

[1] Bryant RE, Katz RH, Lazowska ED. Big-Data Computing: Creating Revolutionary Breakthroughs in Commerce, Science and Society. 2nd ed., Washington: Computing Community Consortium, 2008. 1−15.

[2] Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1):107−113. [doi: 10.1145/1327452.1327492]

[3] Data centers only operating at 4% utilization. 2014. http://www.environmentalleader.com/2010/07/08/data-centers-only-operating-at-4-utilization/

[4] Song J, Li TT, Zhu ZL, Bao YB, Yu G. Benchmarking and analyzing the energy consumption of cloud data management system. Chinese Journal of Computers, 2013,36(7):1485−1499 (in Chinese with English abstract).

[5] Song J, Li TT, Yan ZX, Na J, Zhu ZL. Energy-Efficiency model and measuring approach for cloud computing. Ruan Jian Xue Bao/Journal of Software, 2012,23(2):200−214 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4144.htm [doi: 10.3724/SP.J.1001.2012.04144]

[6] Lin B, Li SS, Liao XK, Meng LB, Liu XD, Huang H. Seadown: SLA-Aware size-scaling power management in heterogeneous mapreduce cluster. Chinese Journal of Computers, 2013,36(5):977−987 (in Chinese with English abstract).

[7] Song J, Hou HY, Wang Z, Zhu ZL. Improved energy-efficiency measurement model for cloud computing. Journal of Zhejiang University (Engineering Science), 2013,47(1):44−52 (in Chinese with English abstract).

[8] Chen YP, Keys L, Katz RH. Towards energy efficient MapReduce. Technical Report, UCB/EECS-2009-109, Berkeley: EECS Department, University of California, 2009. 1−21.

[9] Leverich J, Kozyrakis C. On the energy (in)efficiency of hadoop clusters. ACM SIGOPS Operating Systems Review, 2010,44(1):61−65. [doi: 10.1145/1740390.1740405]

[10] Chen YP, Ganapathi A, Katz RH. To compress or not to compress-compute vs. IO tradeoffs for MapReduce energy efficiency. In: Barford P, Padhye J, Sahu S, eds. Proc. of the 1st ACM SIGCOMM Workshop on Green Networking. New York: ACM Press, 2010. 23−28. [doi: 10.1145/1851290.1851296]

[11] Marinelli EE. Hyrax: Cloud computing on mobile devices using MapReduce [MS. Thesis]. Pittsburgh: Carnegie-Mellon University, 2013.

[12] Yigitbasi N, Datta K, Jain N, Willke T. Energy efficient scheduling of MapReduce workloads on heterogeneous clusters. In: Proc. of the 2nd Int'l Workshop on Green Computing Middleware. Lisbon: Association for Computing Machinery, 2011. 1−6. [doi: 10.1145/2088996.2088997]

[13] Kaushik RT, Bhandarkar M. GreenHDFS: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In: Proc. of the 2010 Int'l Conf. on Power Aware Computing and Systems. Vancouver: USENIX Association, 2010. 1–9.

[14] Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In: Leighton FT, Shor PW, eds. Proc. of the 29th Annual ACM Symp. on the Theory of Computing. New York: ACM Press, 1997. 654−663. [doi: 10.1145/258533.258660]

[15] Liao B, Yu J, Sun H, Nian M. Energy-Efficient algorithms for distributed storage system based on data storage structure reconfiguration. Journal of Computer Research and Development, 2013,50(1):3−18 (in Chinese with English abstract).

[16] Zhang WZ, Zhang HL, Xu X, He H. Distributed search engine system productivity modeling and evaluation. Ruan Jian Xue Bao/ Journal of Software, 2012,23(2):253−265 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4140.htm [doi: 10.3724/SP.J.1001.2012.04140]

[17] Tan YM, Zeng GS, Wang W. Policy of energy optimal management for cloud computing platform with stochastic tasks. Ruan Jian Xue Bao/Journal of Software, 2012,23(2):266−278 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4143.htm [doi: 10.3724/SP.J.1001.2012.04143]

[18] Wang GB, Yang XJ, Tang T, Xu XH. Energy optimization model for heterogeneous parallel system. Ruan Jian Xue Bao/Journal of Software, 2012,23(6):1382−1396 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4078.htm [doi: 10.3724/SP.J.1001.2012.04078]

[19] Elnozahy ENM, Kistler M, Rajamony R. Energy-Efficient server clusters. In: Proc. of the Power-Aware Computer Systems. Berlin, Heidelberg: Springer-Verlag, 2003. 179−197. [doi: 10.1007/3-540-36612-1_12]

[20] Lee KG, Veeravalli B, Viswanathan S. Design of fast and efficient energy-aware gradient-based scheduling algorithms for heterogeneous embedded multiprocessor systems. IEEE Trans. on Parallel and Distributed Systems, 2009,20(1):1−12. [doi: 10.1109/TPDS.2008.55]

[21] Costa GD, Gelas JP, Georgiou Y, Lefevre L, Orgerie AC, Pierson JM, Richard O, Sharma K. The green-net framework: Energy efficiency in large scale distributed systems. In: Proc. of the 23rd IEEE Int'l Symp. on Parallel and Distributed Processing. Washington: IEEE Computer Society, 2009. 1–8. [doi: 10.1109/IPDPS.2009.5160975]

[22] Beloglazov A, Buyya R. Energy efficient resource management in virtualized cloud data centers. In: Proc. of the 10th IEEE/ACM Int'l Conf. on Cluster, Cloud and Grid Computing. Washington: IEEE Computer Society, 2010. 826−831. [doi: 10.1109/CCGRID.2010.46]

[23] Chen T, Xiao N, Liu F, Fu CS. Clustering-Based and consistent hashing-aware data placement algorithm. Ruan Jian Xue Bao/ Journal of Software, 2010,21(12):3175−3185 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3706.htm [doi: 10.3724/SP.J.1001.2010.03706]

[24] Brinkmann A, Salzwedel K, Scheideler C. Efficient, distributed data placement strategies for storage area networks. In: Proc. of the 12th Annual ACM Symp. on Parallel Algorithms and Architectures. New York: ACM Press, 2000. 119−128. [doi: 10.1145/341800.341815]

[25] Zhang W, Song Y, Ruan L, Zhu MF, Xiao LM. Resource management in Internet-oriented data centers. Ruan Jian Xue Bao/Journal of Software, 2012,23(2):179−199 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4146.htm [doi: 10.3724/SP.J.1001.2012.04146]

[26] 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).

[27] Cortes T, Laborta J. A case for heterogeneous disk array. In: Proc. of the IEEE Int'l Conf. on Cluster Computing. New York: ACM Press, 2000. 319−325. [doi: 10.1109/CLUSTR.2000.889085]

[28] Cortes T, Laborta J. Extending heterogeneity to RAID level 5. In: Park Y, ed. Proc. of the 2001 USENIX Annual Technical Conf. on General Track. 2001. 119−132.

[29] Brinkmann A, Effert S, Heide FM, Scheideler C. Dynamic and redundant data placement. In: Proc. of the 27th IEEE Int'l Conf. on Distributed Computing Systems. Washington: IEEE Computer Society, 2007. 29. [doi: 10.1109/ICDCS.2007.103]

[30] WordCount program. Hadoop Source Distribution, 2014. http://github.com/apache/hadoop/blob/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/WordCount.java

[31] Sort program. Hadoop Source Distribution, 2014. http://github.com/apache/hadoop/blob/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/Sort.java

[32] MRBench program. Hadoop Source Distribution, 2014. http://github.com/apache/hadoop/blob/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/MRBench.java

[33] Xu ZC, Tu YC, Wang XR. Exploring power-performance tradeoffs in database systems. In: Li FF, Moro MM, Ghandeharizadeh S, Haritsa JR, Weikum G, Carey MJ, Casati F, Chang EY, Manolescu I, Mehrotra S, Dayal U, Tsotras VJ, eds. Proc. of the 26th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, IEEE, 2010. 485−496. [doi: 10.1109/ICDE.2010.5447840]

[34] Tsirogiannis D, Harizopoulos S, Shah MA. Analyzing the energy efficiency of a database server. In: Elmagarmid AK, Agrawal D, eds. Proc. of the ACM Int'l Conf. on Management of Data. Indianapolis: ACM Press, 2010. 231−242. [doi: 10.1145/1807167.1807194]

附中文参考文献:

[4] 宋杰,李甜甜,朱志良,鲍玉斌,于戈.云数据管理系统能耗基准测试与分析.计算机学报,2013,36(7):1485−1499.

[5] 宋杰,李甜甜,闫振兴,那俊,朱志良.一种云计算环境下的能效模型和度量方法.软件学报,2012,23(2):200−214. http://www.jos.org.cn/1000-9825/4144.htm [doi: 10.3724/SP.J.1001.2012.04144]

[6] 林彬,李姗姗,廖湘科,孟令丙,刘晓东,黄詠.Seadown:一种异构 MapReduce 集群中面向 SLA 的能耗管理方法.计算机学报,2013,36(5):977−987.

[7] 宋杰,侯泓颖,王智,朱志良.云计算环境下改进的能效度量模型.浙江大学学报(工学版),2013,47(1):44−52.

[15] 廖彬,于炯,孙华,年梅.基于存储结构重配置的分布式存储系统节能算法.计算机研究与发展,2013,50(1):3−18.

[16] 张伟哲,张宏莉,许笑,何慧.分布式搜索引擎系统能效建模与评价.软件学报,2012,23(2):253−265. http://www.jos.org.cn/1000-9825/4140.htm [doi: 10.3724/SP.J.1001.2012.04140]

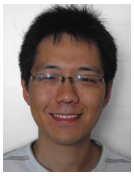
[17] 谭一鸣,曾国荪,王伟.随机任务在云计算平台中能耗的优化管理方法.软件学报,2012,23(2):266−278. http://www.jos.org.cn/1000-9825/4143.htm [doi: 10.3724/SP.J.1001.2012.04143]

[18] 王桂彬,杨学军,唐滔,徐新海.异构并行系统能耗优化分析模型.软件学报,2012,23(6):1382−1396. http://www.jos.org.cn/1000-9825/4078.htm [doi: 10.3724/SP.J.1001.2012.04078]

[23] 陈涛,肖侬,刘芳,付长胜.基于聚类和一致 Hash 的数据布局算法.软件学报,2010,21(12):3175−3185. http://www.jos.org.cn/1000-9825/3706.htm [doi: 10.3724/SP.J.1001.2010.03706]

[25] 张伟,宋莹,阮利,祝明发,肖利民.面向 Internet 数据中心的资源管理.软件学报,2012,23(2):179−199. http://www.jos.org.cn/1000-9825/4146.htm [doi: 10.3724/SP.J.1001.2012.04146]

[26] 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.



宋杰(1980－),男,安徽淮北人,博士,副教授,CCF 高级会员,主要研究领域为高能效计算,海量数据计算,云计算.

李甜甜(1989－),女,博士生,CCF 学生会员,主要研究领域为高能效计算.

王智(1991－),男,硕士,主要研究领域为高能效计算.

于戈(1962－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为数据库理论和技术,分布与并行系统.