不同 MapReduce 运行系统的性能测试与分析

易修文 李天瑞 张钧波 滕 飞
(西南交通大学信息科学与技术学院 成都 610031)

摘 要 随着云计算技术的发展,许多 MapReduce 运行系统被开发出来,如 Hadoop、Phoenix 和 Twister 等。直观上,Hadoop 具有很强的可扩展性、稳定性,适合处理大规模离线应用;Phoenix 具有运行速度快等优点,适合处理数据密集型任务;Twister 是轻量级的迭代系统,非常适合迭代式的应用。不同的应用在不同的 MapReduce 运行系统中有着不同的性能。通过测试不同应用在这些运行系统上的性能,给出了实验比较和性能分析,从而为大数据处理时选择合适的并行编程模型提供依据。

关键词 云计算,MapReduce,Hadoop,Phoenix,Twister
中图法分类号 TP316.4 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.5.005

Performance Testing and Analysis among Different MapReduce Runtime Systems

YI Xiu-wen LI Tian-rui ZHANG Jun-bo TENG Fei
(School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China)

Abstract With the development of cloud computing technology, several implementations which adopt MapReduce model, e.g., Hadoop, Phoenix and twister, have been developed. Hadoop has high scalability and stability, thus is suitable for handling large-scale off-line applications. The primary advantage of Phoenix, which is especially appropriate for data-intensive tasks, is its processing speed. Twister, a lightweight iterative runtime system, is designed for iterative applications. Different applications produce different levels of performance on different MapReduce runtime systems. By testing various applications using the aforementioned runtime systems, the experimental comparison and performance analysis were presented, providing the basis for the selection of parallel programming models for big data processing.

Keywords Cloud computing, MapReduce, Hadoop, Phoenix, Twister

1 引言

随着信息技术的快速发展,在信息检索和天文学等众多领域,每天都有大量的数据生成,进而形成了大数据。如何从这些数据中快速有效地挖掘出有用信息成为一项新的挑战,因此当前迫切需要的是寻求一个方法来高效处理这些大数据[1]。

云计算[2]通过互联网连接,为用户提供可升级的计算资源和服务。它融合了网格计算、分布式计算和并行计算等特点,具有超强的运算能力,能够以较低的成本处理大规模数据。目前,在云计算技术中,基于大规模计算机集群的分布式并行编程和基于多核/多线程的并发编程已经越来越流行,在多个领域中均得到成功应用。

MapReduce[3]是云计算的核心技术之一,是 Google 提出的一个用于大规模数据集并行运算的软件架构。这个框架解决了诸如机器间通信、任务调度和数据处理等疑难问题。它具有良好的分布性和扩展性,人们可以快速地搭建 MapReduce 的实现平台,以在大规模计算集群上使用。它有效降低了并行编程的难度,没有分布式并行编程经验的程序员们也可以很轻松地在 MapReduce 的实现平台上运行大规模数据处理的程序。

随着 MapReduce 编程模型研究的深入,许多 MapReduce 运行时系统被开发出来,它们各自有着不同的体系结构和实现目的。例如,(1)Hadoop[4]是一个适合处理数据密集型应用的开源框架,具有很强的可扩展性、稳定性。它可以很方便地移植到亚马逊 EC2 平台和 Windows Azure 系统上,分别被叫做 Amazon Elastic MapReduce 和 HadoopOnAzure;(2)Phoenix[5]是基于多核/多线程系统开发的 MapReduce 运行系统,以实现共享内存为目的,具有运行速度快等优点;(3)Twister[6]是轻量级的 MapReduce 运行系统,非常适合迭代式的应用,有一个在 Windows Azure 实现的 Twister 平台叫 Twister4Azure;(4)Mars[7]是在图形处理器(Graphic Processing Unit,GPU)上开发的 MapReduce 框架,它通过简单熟悉的 MapReduce 接口隐藏了 GPU 编程的复杂性,降低了 GPU

到稿日期:2014-06-17 返修日期:2014-09-25 本文受国家自然科学基金(61175047,61202043),国家自然科学基金联合基金(U1230117),四川省科技支撑计划项目(2012RZ0009),中央高校基本科研业务费专项资金(SWJTU11ZT08)资助。

易修文(1991-),男,硕士生,主要研究领域为云计算、数据挖掘,E-mail:xiuwenyi@gmail.com;**李天瑞**(1969-),男,博士,教授,博士生导师,主要研究领域为数据挖掘、粗糙集、粒计算和云计算;**张钧波**(1986-),男,博士生,主要研究领域为云计算、数据挖掘、粒计算和粗糙集;**滕 飞**(1984-),女,博士,讲师,主要研究领域为云计算、调度和资源优化。

编程的难度；(5)Spark[8]是一个类 MapReduce 的并行计算框架，它将数据尽可能地放到内存中，适合用于低延迟的迭代计算和交互式分析。

文献[9-11]均只从理论方面比较了几个 MapReduce 实现平台，讨论了它们各自的系统架构、原理机制、功能特性以及适用范围等，但都没有进行实验验证。本文选取了 3 个典型 MapReduce 实现平台，分别为使用最为广泛的 Hadoop 平台、基于共享内存的 Phoenix 平台和适合迭代的 Twister 平台。围绕这 3 个平台，设计了几组不同平台之间的对比实验，以测试 WordCount、K-means 和 Matrix multiply 这 3 个算法在这些运行时系统上的性能。通过分析实验数据和比较不同平台的运行结果，得出不同平台在不同应用上的优缺点，并总结出不同平台的特性，从而给出它们各自的适用范围，进而为处理大数据时选择合适的并行编程模型提供参考。

2 预备知识

2.1 MapReduce

MapReduce 提供了一个简便的方法将数据并行处理，隐藏了底层实现细节，有效地降低了并行编程的难度，简单易学。它的编程思想源自于函数式编程语言 Lisp。基于“分而治之”的思想，MapReduce 将复杂的并行计算过程高度地抽象为两个函数：Map(映射)和 Reduce(化简)，简单地说就是“任务的分解与结果的汇总”。用户只需要实现 Map 和 Reduce 函数，而不需要关注数据是如何切分的以及任务是怎么分配的。

MapReduce 编程模型的基本流程如图 1 所示。Map 函数接收一个输入的〈key，value〉对，然后产生一个中间的〈key，value〉对。MapReduce 把所有具有相同中间 key 值的中间 value 值集合在一起，再传递给 Reduce 函数。Reduce 函数接收输入后，合并这些 value 值并形成一个 value 值的集合，最后计算得出最终结果并输出。

图 1 MapReduce 的基本流程

2.2 Hadoop

Apache 软件基金会开发的 Hadoop 是一个用于处理大规模数据集、面向集群环境的分布式开源软件框架，目前已发展到 2. x 版本。2. x 在 1. x 版本上有了很多改进，最核心的是在 YARN 框架上提供了一些可以运行的计算框架，如 Storm、Giraph 等。本文主要讨论传统的 1. x 版本 Hadoop。HDFS[12](Hadoop Distributed File System)是一种专门为大规模分布式数据处理而设计的分布式文件系统，它提供了数据存储与访问，默认数据块大小是 64MB。Hadoop MapReduce 由一个 Jobtracker 和多个 Tasktracker 组成，Jobtracker 主要负责调度所有作业，Tasktracker 则执行用户写的 Map 和 Reduce 函数。Hadoop 的 MapReduce 作业主要包括以下几个步骤：提交作业、初始化作业、分配任务、执行任务、更新进度和状态、完成。与 Hadoop 相关的项目还有很多，如 Hive[13]、Pig[14]、HadoopDB[15]等。Mahout[16]是一个 Apache 的开源机器学习和数据挖掘项目，提供了一些可扩展的机器学习领域经典算法的实现。

2.3 Phoenix

美国斯坦福大学实现的 Phoenix 是一个在多核平台上实现的专门处理数据密集型任务的 MapReduce 计算框架，采用 C、C++语言编写，具有运行速度快的优点。它以实现共享内存为目的，降低了由数据间通信和任务调度等所产生的影响，从而使程序执行更加高效。Phoenix 是建立在 P-thread 基础上的，可以方便地移植到其它共享内存线程库上。它利用共享内存缓冲区实现通信，从而避免了因数据拷贝产生的开销。Phoenix 由一组对程序应用开发者开放的简单 API 和 1 个高效的运行时组成，提供了资源调控、并发管理和故障恢复等功能。在计算机配置限定下，线程数量越多，程序运行的速度也越快。

2.4 Twister

美国印第安纳大学开发的 Twister 是一个开源的、轻量级的 MapReduce 运行系统，利用流处理技术为迭代任务提供高效的计算框架。与典型的 MapReduce 运行系统设计不同，它没有底层的分布式文件系统，而采用发布/订阅的消息传递机制来实现通信和数据传输。它从工作节点的本地磁盘读取数据，在工作节点的分布式内存中处理中间数据，数据传输不需要经过额外的磁盘读写，这就使性能有了极大的提升。同时，它支持长时间运行的 Map/Reduce，配置一次，可多次使用。Twister 没有提供针对单个节点的容错机制，但可以保存迭代的计算状态，一旦出现故障，整个计算可以回滚几个迭代后继续进行。

3 实验设计

本文所使用的集群环境由 5 台计算机组成(1 个 Master 节点，4 个 Slave 节点)，系统为 Ubuntu12. 04，64 位。Master 节点的计算机配置为 2 核，主频 2. 53GHz，内存 4GB。4 个 Slave 节点计算机的配置均为 2 核，主频 2. 20GHz，内存 2GB。Hadoop 平台由 1 个 Master 节点和 4 个 Slave 节点组成，版本为 Hadoop1. 2. 0；Phoenix 平台安装在 Master 节点下，单机，版本为 Phoenix++；Twister 平台由 5 个节点组成，Master 节点作 Broker Network，版本为 Twister0. 9。还有一台多核计算机作为多线程实验的机器，48G 内存，2. 40GHz 的主频，16 核，安装了 Phoenix++。

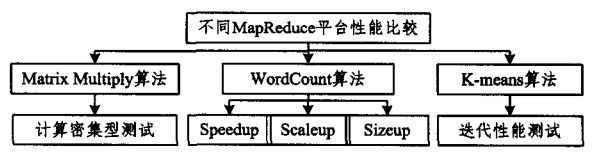

图 2 实验设计

为给出不同 MapReduce 平台的性能比较，设计了几个实验，如图 2 所示。不同的算法在不同的平台上有着不同的表现，采用 WordCount、K-means 和 Matrix Multiply 3 个算法来进行测试，将可能出现的问题呈现出来。WordCount 算法是用来统计一系列字符型文件中每个单词出现的频率，实验具体围绕并行算法的 3 个指标 Speedup、Scaleup 和 Sizeup 展开。K-means 算法是聚类分析中使用最为广泛的算法之一，用于测试不同平台的迭代性能。Matrix Multiply 是典型的计

算密集型应用，输入的数据量不大，但计算量非常大。最后还采用 WordCount 算法在 Phoenix 平台下做了多线程实验，该实验运行在多核计算机上。所有实验所用数据均由 Twister 平台产生。

4 实验比较

4.1 Speedup

Speedup 是指同一个任务在单处理器系统和并行处理器系统中运行消耗的时间比率。计算 Speedup 的方法是：保持数据大小不变，增加计算机的数目。其中，p 是节点的数量，T_1 是在 1 个节点参与实验的情况下的运行时间，T_p 是在 p 个节点下的运行时间。具体如下：

$$Speedup(p)=\frac{T_1}{T_p} \tag{1}$$

第一个实验是测试在不断增加节点数时，WordCount 算法在 Hadoop 和 Twister 这两个平台上分别运行所需的时间。实验所用的节点数从 1 增加到 5，数据集大小为 4G。实验结果如图 3 所示。

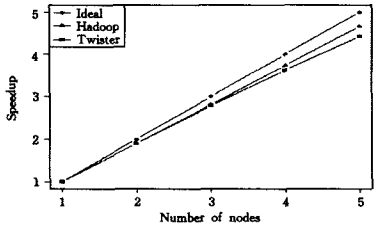

图 3 Speedup

Speedup 指标每条线的斜率表明了算法的加速比，即算法的并行程度。随着工作节点数的增加，Hadoop 和 Twister 的 Speedup 均保持线性增长。这说明了两个平台对于 WordCount 算法均有比较好的并行效果，同时也说明 WordCount 算法非常适合并行。

由于实验环境所限，只测试了 5 台计算节点的并行。虽然实验中所示的 Speedup 指标还是比较理想，但线性的 Speedup 通常是难以达到的。随着节点数的不断增加，加速比的增加速度会不断变慢，这是因为节点间的通讯损耗问题凸显出来，还有就是各个计算节点本身的问题，比如算法所花费的总时间通常是由最慢的机器决定，这样就导致线性的 Speedup 在实际应用中很难实现。

4.2 Scaleup

Scaleup 同样也是衡量并行性能的一个重要指标，指当系统的节点数和数据集大小呈相当水平的增长时，算法运行消耗的时间比率。计算 Scaleup 的方法是，扩大数据的同时增加计算机的数目。具体如下：

$$Scaleup(D,p)=\frac{T_{D_1}}{T_{D_p}} \tag{2}$$

其中，D 是数据集，T_{D_1} 是一个节点上数据集 D 的执行时间，T_{D_p} 是 p 个节点上数据集 $p\times D$ 的执行时间。

第二个实验是测试在不断增加节点数时，同比例增大数据集，WordCount 算法在 Hadoop 和 Twister 这两个平台上分别运行所需要的时间。节点数从 1 增加到 5，数据集大小同比例从 1G 增加到 5G。实验结果如图 4 所示。

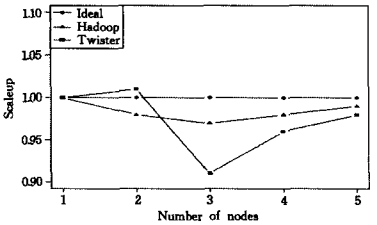

图 4 Scaleup

随着节点数的增加，两个平台的 Scaleup 指标都在 1.0 附近浮动，这表明大数据任务可以通过增加计算节点数来抵消计算时间的增长，同时也表明 Hadoop 和 Twister 平台都具有比较好的可扩展性。但 Twister 曲线上下波动较大，相对而言，Hadoop 更加稳定。

4.3 Sizeup

Sizeup 是用来测试算法本身的一个时间复杂度，反映了数据集不断增加的情况下并行算法的性能。计算 Sizeup 的方法是保持计算机的数目不变，增大实验所用数据集的大小。具体如下：

$$Sizeup(p)=\frac{T_{S_p}}{T_{S_1}} \tag{3}$$

其中，T_{S_p} 是数据集 $p\times D$ 的执行时间，T_{S_1} 是数据集 D 的执行时间。

第三个实验是在不断增大数据集时，WordCount 算法在 Hadoop、Phoenix 和 Twister 这 3 个平台上运行分别所需的时间。其中，Hadoop 和 Twister 平台都只用 3 个计算节点，Phoenix 单机。数据集的大小从 1G 逐渐增加到 32G，3 个平台的 Sizeup 曲线如图 5 所示，具体运行时间如表 1 所列。

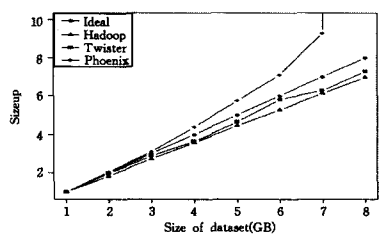

图 5 Sizeup

表 1 不同平台处理不同大小的数据集的时间(单位：s)

数据集大小	Hadoop	Twister	Phoenix
1GB	182	43	20
2GB	334	85	41
4GB	647	157	88
8GB	1271	314	崩溃
16GB	2603	862	—
32GB	5589	崩溃	—

从整体上看，3 个平台均呈现一个较好的线性增长。其中，Hadoop 的 Sizeup 曲线是最为线性的，可见 Hadoop 具有良好的稳定性。但当运行相同大小的数据集时，Hadoop 所用的时间一直是最多的。当运行 32G 以上数据时，其他平台都已经崩溃，而 Hadoop 却能继续正常运行，这时它的优势才体现出来，说明其更适合处理大数据集。Twister 的运行速度一直在 Hadoop 和 Phoenix 之间。当运行 32G 的数据集时，程序崩溃。Phoenix 平台的 Sizeup 曲线在 Ideal 曲线上方，说明它的稳定性较差，且它只有 7 个数据点，一旦数据集大小超过

8G，程序就发生崩溃；但它的运行时间一直是最短的，具有运行速度快的优点。

4.4 计算密集型比较

实验四是测试 Matrix multiply 算法在 3 个平台上的运行情况。A、B 矩阵的大小相同，均为 1000×1000。实验数据如表 2 所列。

表 2 不同平台运行 Matrix multiply 的时间(单位：s)

矩阵大小	Hadoop	Twister	Phoenix
A&B:1000×1000	892.5	2.1	1.3

Hadoop 平台是基于文本的处理方式，一个 1000×1000 矩阵的文本大小只有 2.1MB，但会产生数个 G 大小的中间结果。这些数据在读写、传输等过程有一定的时间消耗，这就导致了 Hadoop 平台运行速度比较慢。Phoenix 平台花费的时间是最少的，它将所有数据存储在内存中，且在内存中计算，没有其他时间开销，所以它的运行速度是最快的。Twister 平台也是在分布式内存中处理数据，时间花费较少。对于数据集相对较小且计算量较大的应用，与 Hadoop 相比，Twister 的处理速度快；与 Phoenix 相比，它能处理更大的数据。

4.5 迭代性能比较

实验五是测试 K-means 算法在 Hadoop、Phoenix 和 Twister 3 个平台的运行情况，其中 K-means 算法运行在 Mahout 上。实验数据如表 3 所列。

表 3 不同平台运行 K-means 的时间(单位：s)

数据集大小	Hadoop	Twister	Phoenix
36000 个二维数据点	648.9	3.4	0.2

Hadoop 的运行速度依然很慢，这是由于 Hadoop 不支持迭代计算且花费了大量时间在数据通信上，大大地降低了执行速度。Phoenix 则表现得最为高效，对于内存可以负载的数据集，运行速度特别快。Twister 平台的运行时间远远少于 Hadoop，这是因为 Twister 支持长时间运行的 Map/Reduce 任务。在每次迭代中，分布式内存缓存中间结果，大大降低了读写文件的开销，对于迭代应用有较好的性能表现。

4.6 多线程比较

实验六是测试当不断增加线程数量时，Phoenix 平台运行 WordCount 算法处理 1G、2G、3G 数据的时间。具体实验的数据如表 4 所列。

表 4 不同线程下 Phoenix 运行 WordCount 的时间(单位：s)

数据集大小	1 个线程	2 个线程	4 个线程	8 个线程	16 个线程	32 个线程
1G	13.490	7.162	3.996	2.359	1.884	1.889
2G	26.793	14.128	8.046	4.786	3.766	3.783
3G	40.134	21.182	12.078	7.027	5.610	5.658

纵向地看，数据集大小成比例增加时，运行时间也成比例增加，这说明 Phoenix 平台有着比较好的稳定性。横向看，随着线程数的不断增多，直到 16 个线程，Phoenix 运行时间不断减少，可见其是基于多线程机制的运行系统。随着线程数的不断增多，直到 16 个线程，运行时间减少的比率也在不断减小，从 1 个线程到 2 个线程降低得最为明显，后面则越来越不明显。用 32 个线程时，运行时间反而比 16 个线程的多，比 8 个线程的少。这是因为实验所用的计算机只有 16 核，即最多只能有 16 个线程起作用，当设置的线程数量多于 16 时，Phoenix 花费了一些时间来调度其他无效的线程，反而减少了程序的运行时间。由此可见，线程数的增多可以加快运行速度，但有瓶颈。

5 结果分析

根据以上的实验比较可以很清楚地知道，相同的应用在不同的 MapReduce 运行时系统上所花费的时间是不一样的。对于 Hadoop 平台，本地数据首先会上传到 HDFS 上，HDFS 自动对数据进行切分；然后默认调度器(FIFO)开始调度，Map 和 Reduce 相继开始工作。Map 过程产生的中间数据首先写到本地磁盘上，Reduce 过程又从本地磁盘读取数据，中间数据的读写、传输极大增加了 Hadoop 的运行时间，即使输入数据不大，也会花费几秒时间。因此，Hadoop 上的程序运行速度相对较慢。但正是由于这种基于磁盘的处理方式，使得 Hadoop 反而适合处理大规模的数据集。

相反，Twister 和 Phoenix 都没有分布式文件系统。对于 Twister 平台，用户需手动借助一个脚本来切分数据并将数据发送到各个计算节点上，同时产生一个包含数据存放位置的配置文件，Twister 会根据这个配置文件来处理数据。与 Hadoop 平台不同，Twister 在工作节点的分布式内存中处理中间数据且采用了发布/订阅的消息传递机制，这样不仅极大地提高了数据传输与处理的效率，还节省了很多磁盘空间。Twister 依赖于分布式内存，一旦数据集过大导致内存难以负荷，程序就将发生崩溃。

对于 Phoenix 平台，所有的输入数据全部读到内存中，在内存中可直接进行计算，大大避免了文件多次读写的开销；而且它只支持单机，不需要与其他节点进行通信，没有通信上的花费。Phoenix 是用 C++语言编写的，运行速度自然比采用 JAVA 编写的 Hadoop 和 Twister 快得多。Phoenix 是建立在 P-thread 基础上的，可用线程数越多，程序的并行度也越高，运行速度也越快。因此，只要输入数据能载入到内存，Phoenix 的性能都是最好的。

结束语 不同平台的原理机制不同，导致它们各自的性能和适用范围也不同。Hadoop 平台适用于数据密集型应用的计算，具有很强的可扩展性、稳定性，但对小规模数据的效率不高，单机性能差。Phoenix 平台具有运行速度快的优点，适合用于处理计算密集型的应用，尤其适合处理小量的数据集，但受限于单机物理内存，无法处理海量数据。Twister 平台适合用于迭代计算，对于图算法有很高的适用性，它的运行速度相对较快，但依赖集群内存大小。

由于实验条件所限，目前只测试了 5 个计算节点的并行，而且本文的实验所用计算节点的配置不高。本文只分析比较了上述 3 种平台，后续的工作可以考虑比较其他的 MapReduce运行系统，如 Spark、Mars 等。

参 考 文 献

[1] Pan Y, Zhang J B. Parallel Programming on Cloud Computing Platforms: Challenges and Solutions[J]. KITCS/FTRA Journal of Convergence, 2012, 3(4): 23-28

(下转第 50 页)

惯更一致；而区域增长和阈值方法更容易受周围噪声的干扰。

结束语 本文实现了日冕暗化的自动检测和提取工作。通过分块地进行数据分析，增强了暗化的统计特征，利用 Adaboost 分类方法对暗化进行检测，获得了一定效果。将随机游走方法和检测的结果相结合实现自动提取暗化区域，可以克服区域增长和阈值方法容易受周围噪声干扰的缺点，能获得更加完整、连续的日冕暗化区域。目前的数据集相对较小，后期将扩大数据测试算法效果。

由于本文的暗化检测和提取建立在 BD 图像的基础上，基准图的选择显得十分重要，否则容易受其它现象的干扰。如何合理选择基准图是今后需要完善的工作。

参 考 文 献

[1] Harrison R A, Lyons M. A spectroscopic study of coronal dimming associated with a coronal mass ejection[J]. Astronomy and Astrophysics, 2000(358): 1097-1108

[2] Hudson H S, Lemen J R, Webb D F. Coronal X-Ray Dimming in Two Limb Flares[C]// Proceedings of a Yohkoh Conference. 1996: 379-382

[3] Gopalswamy N, Hanaoka Y, Hudson H S. Structure and dynamics of the corona surrounding an eruptive prominence[J]. Advances in Space Research, 2000, 25(9): 1851-1854

[4] Hudson H S, Lemen J R, Cyr O C S, et al. X-ray coronal changes during halo CMEs[J]. Geophysical Research Letters, 1998, 14(25): 2481-2484

[5] Delaboudinière J P, Artzner G E, Brunaud J, et al. EIT: Extreme-ultraviolet Imaging Telescope for the SOHO mission[J]. Solar Physics, 1995, 162(1/2): 291-312

[6] Podladchikova O, Berghmans D. Automated Detection of Eit Waves And Dimmings[J]. Solar Physics, 2005, 228(1/2): 265-284

[7] Attrill G D R, Wills-Davey M J. Automatic Detection and Extraction of Coronal Dimmings from SDO/AIA Data[J]. Solar Physics, 2010, 262(2): 461-480

[8] Attrill G, Nakwacki M S, Harra L K, et al. Using the Evolution of Coronal Dimming Regions to Probe the Global Magnetic Field Topology[J]. Solar Physics, 2006, 238(1): 117-139

[9] Reinard A A, Biesecker D A. Coronal Mass Ejection-Associated Coronal Dimmings[J]. The Astrophysical Journal, 2008, 674(1): 576

[10] Krista L D, Reinard A. Study of the Recurring Dimming Region Detected at AR 11305 Using the Coronal Dimming Tracker (CoDiT) [J]. The Astrophysical Journal, 2013, 762(2): 91

[11] Ayinala M, Parhi K K. Low complexity algorithm for seizure prediction using Adaboost[C]// Proceedings of the Engineering in Medicine and Biology Society (EMBC). San Diego, 2012: 1061-1064

[12] Cao J, Kwong S, Wang R. A noise-detection based AdaBoost algorithm for mislabeled data[J]. Pattern Recognition, 2012, 45(12): 4451-4465

[13] Lan R S, Jiang Y, Ding L G, et al. Automated flare prediction using the AdaBoost algorithm[J]. Research in Astronomy and Astrophysics, 2012, 12(9): 1191-1196

[14] Freeland S L, Handy B N. Data Analysis with the SolarSoft System[J]. Solar Physics, 1998, 182(2): 497-500

[15] Grady L. Multilabel random walker image segmentation using prior models[C]// Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Volume 1, San Diego, 2005: 763-770

(上接第 27 页)

[2] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing [J]. Communications of the ACM, 2010, 53(4): 50-58

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

[4] White T. Hadoop: the definitive guide(2nd ed)[M]. O'Reilly, 2012

[5] Talbot J, Yoo R M, Kozyrakis C. Phoenix++: modular MapReduce for shared-memory systems[C]// Proceedings of the Second International Workshop on MapReduce and its Applications. ACM, 2011: 9-16

[6] Ekanayake J, Li H, Zhang B, et al. Twister: a runtime for iterative mapreduce[C]// Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, 2010: 810-818

[7] He B, Fang W, Luo Q, et al. Mars: a MapReduce framework on graphics processors[C]// Proceedings of the 17th international conference on Parallel architectures and compilation techniques. ACM, 2008: 260-269

[8] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]// Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2

[9] Kang L Y, Wang X Y, Bai R J. Analysis of MapReduce Principle and Its Main Implementation Platforms[J]. New Technology of Library and Information Service, 2012(02): 60-67

[10] Li J J, Li Q, Tian B. The Analysis and Comparison between MapReduce Implementations[J/OL]. http://www.paper.edu.cn/html/releasepaper/2011/11/464. 2011

[11] Li J J, C J, W D, et al. Survey of MapReduce Parallel Programming Model[J]. ACTA Electronica Sinica, 2011(11): 2635-2642

[12] Borthakur D. The hadoop distributed file system: Architecture and design[OL]. http://hadoop.apache.ogr/docs/ro.18.1/hdsf_design.html

[13] Thusoo A, Sarma J S, Jain N, et al. Hive: a warehousing solution over a map-reduce framework[J]. Proceedings of the VLDB Endowment, 2009, 2(2): 1626-1629

[14] Olston C, Reed B, Srivastava U, et al. Pig latin: a not-so-foreign language for data processing[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. ACM, 2008: 1099-1110

[15] Abouzeid A, Bajda-Pawlikowski K, Abadi D, et al. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 922-933

[16] Anil R, Dunning T, Friedman E. Mahout in action[M]. Manning publications, 2011