MapReduce 并行编程模型研究综述

李建江[1],崔 健[1],王 聃[1],严 林[1],黄义双[2]
(1.北京科技大学计算机与通信工程学院计算机科学与技术系,北京 100083;
2.中国石油化工股份有限公司勘探南方分公司研究院,四川成都 610041)

摘 要: MapReduce 并行编程模型通过定义良好的接口和运行时支持库,能够自动并行执行大规模计算任务,隐藏底层实现细节,降低并行编程的难度.本文对 MapReduce 的国内外相关研究现状进行了综述,阐述和分析了当前国内外与 MapReduce 相关的典型研究成果的特点和不足,重点对 MapReduce 涉及的关键技术(包括:模型改进、模型针对不同平台的实现、任务调度、负载均衡和容错)的研究现状进行了深入的分析.本文最后还对 MapReduce 未来的发展趋势进行了展望.

关键词: MapReduce; 并行编程模型; 运行时支持库; 海量数据处理
中图分类号: TP316.4 **文献标识码:** A **文章编号:** 0372-2112 (2011) 11-2635-08

Survey of MapReduce Parallel Programming Model

LI Jian-jiang[1], CUI Jian[1], WANG Dan[1], YAN Lin[1], HUANG Yi-shuang[2]
(1. *Department of Computer Science and Technology, School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing* 100083, *China*; 2. *Research Institute of Exploration Southern Division Company, SINOPEC, Chengdu, Sichuan* 610041, *China*)

Abstract: Through well-defined interfaces and runtime support library, MapReduce parallel programming model can automatically perform the large-scale computing tasks in parallel, hide the underlying implementation details, and reduce the difficulty of parallel programming. This paper reviews the domestic and overseas research of the MapReduce, describes and analyzes the characteristics and lack of the typical research achievements about MapReduce at home and abroad. Then this paper focus on the in-depth analysis of the key technologies about MapReduce (including: model optimization, model implementation according to the different platforms, task scheduling, load balancing, and fault tolerance). Finally, this paper prospects the MapReduce for the future trend.

Key words: MapReduce; parallel programming model; runtime support library; massive data processing

1 引言

MapReduce[1]是 Google 公司于 2004 年提出的能并发处理海量数据的并行编程模型,其特点是简单易学、适用广泛,能够降低并行编程难度,让程序员从繁杂的并行编程工作中解脱出来,轻松地编写简单、高效的并行程序.

传统并行编程模型可分为两类:数据并行模型和消息传递模型.其中,数据并行模型的典型代表是 HPF[2],消息传递模型的典型代表是 MPI[3]和 PVM[4].数据并行模型级别较高,编程相对简单,但是仅适用于解决数据并行问题.使用消息传递模型编写并行程序时,用户需要显式进行数据与任务量的划分、任务之间的通信与同步、死锁检测[5]等,编程负担较重.

针对上述问题,MapReduce 并行编程模型的最大优势在于能够屏蔽底层实现细节,有效降低并行编程难度,提高编程效率.其主要贡献在于:①使用廉价的商用机器组成集群,费用较低,同时又能具有较高的性能;②松耦合和无共享结构使之具有良好的可扩展性;③用户可根据需要自定义 Map、Reduce 和 Partition 等函数;④提供了一个运行时支持库,它支持任务的自动并行执行.提供的接口便于用户高效地进行任务调度、负载均衡、容错和一致性管理等;⑤MapReduce 适用范围广泛,不仅适用于搜索领域,也适用于满足 MapReduce 要求的其它领域计算任务.

2 MapReduce 总体研究状况

最近几年,在处理 TB 和 PB 级数据方面,MapReduce 已经成为使用最为广泛的并行编程模型之一.国内外 MapReduce 相关的研究成果主要有以下几方面:

收稿日期:2011-01-12;修回日期:2011-03-17
基金项目:教育部重点基金(No.108008);国家 863 高技术研究发展计划(No.2008AA01Z109);北京市教育重点学科计算机系统结构(No.XK100080537)

(1)在编程模型改进方面:MapReduce 存在诸多不足.目前,典型研究成果有 Barrier-less MapReduce[6]、MapReduceMerge[7]、Oivos[8]、Kahn process networks[9]等.但这些模型均仅针对 MapReduce 某方面的不足,研究片面,并且都没有得到广泛应用,部分模型也不成熟.

(2)在模型针对不同平台的实现方面:典型研究成果包括:Hadoop[10]、Phoenix[11,12]、Mars[13]、Cell MapReduce[14]、Misco[15]和 Ussop[16].部分平台(例如:GPUs 和 Cell/B.E.)由于底层硬件比较复杂,造成编程难度较大,增加了用户编程的负担.

(3)在运行时支持库(包括:任务调度、负载均衡和容错)方面:常用的任务调度策略是任务窃取,但该策略有时会加大通信开销.典型的研究成果包括:延迟调度策略[17]、LATE 调度策略[18]和基于性能驱动的任务调度策略[19]等.在容错方面的典型研究成果是 reduce 对象[20].目前,运行时支持库中针对一致性管理和资源分配等方面的研究相对较少.

(4)在性能分析与优化方面:目前,文献[21]主要研究在全虚拟环境下 MapReduce 性能分析,文献[22]则提出了名为 MRBench 的性能分析评价指标.性能优化典型成果包括:几何规划[23]、动态优先级管理[24]和硬件加速器[25]等.着眼于性能,结合运行时支持库,将是 MapReduce 研究的热点之一.

(5)在安全性和节能方面:安全性方面典型研究成果是 SecureMR 模型[26].文献[27]和文献[28]则在节能方面做了相应的研究.目前国内外在安全性和节能方面的研究成果相对较少,但是这方面研究的重要性已经得到了越来越多的重视.如果一个模型没有很高的安全性,同时也没有很好地考虑功耗问题,那对其大范围推广将产生致命的影响.

(6)在实际应用方面:MapReduce 应用范围广泛,Google 等诸多公司都在使用 MapReduce 来加速或者简化各自公司的业务[29].MapReduce 还广泛应用于云计算[30]和图像处理[31]等领域.随着科技的进步,MapReduce 将会得到越来越广泛的应用.

国内学者 MapReduce 相关研究成果主要集中在实际应用方面.例如,把 MapReduce 应用于模式发现[32]和数据挖掘[33]等领域.部分研究成果涉及模型针对不同平台的实现、任务调度、容错和性能评估优化.例如,文献[34]提出了名为 FPMR 的基于 FPGA 平台的 MapReduce 实现,文献[35]提出了基于已知数据分布的任务调度策略,文献[36]提出了名为 SAMR 的异构环境下自适应任务调度策略,文献[37]提出了基于目录的双阶段错误恢复机制,文献[38]提出了名为 The HiBench Benchmark Suite 的性能评估指标,文献[39]提出利用分布式内存缓冲来进行性能优化.综上,国内针对 MapReduce 的研究起步稍晚,绝大部分研究集中在应用方面.对 MapReduce 关键技术也进行了研究.但是相对于国外,国内在这些方面的研究成果较少.

3 MapReduce 模型及其改进

Google 公司最早提出了 MapReduce 并行编程模型,当用户程序调用 MapReduce 运行时支持库时,其执行过程如下(如图 1 所示):

(1)首先,用户程序把输入数据分割成 M 份,每份为 16MB 到 64MB 大小的数据块(可通过参数来设定).然后,开始在集群上进行程序的拷贝.这些程序拷贝中有一份是 Master,其余都是向 Master 请求任务的 Worker;

(2)一旦分配到 Map 任务,Worker 便从相应的输入数据中分析出 key/value 对,并把每个 key/value 对作为用户定义的 Map 函数的输入.Map 函数产生的中间值 key/value 对被存储在内存中;

(3)存储在内存里的中间值 key/value 对会被定期写入本地磁盘中,用户定义的 Partition 函数将其划分为多个部分,Master 负责把这些中间值 key/value 对在本地磁盘上的存储位置传送给执行 Reduce 任务的 Worker;

(4)通过远程过程调用,执行 Reduce 任务的 Worker 从执行 Map 任务的 Worker 的本地磁盘读取中间值 key/value 对;

(5)当一个执行 Reduce 任务的 Worker 从远程读取到所有所需的中间值 key/value 对之后,通过排序将具有相同 key 的中间值 key/value 对聚合在一起,形成 key/values 对,作为 Reduce 函数的输入;

(6)每个 Reduce 函数的输出分别放到相应的输出文件中.当所有的 Map 和 Reduce 任务都执行完毕,Master 唤醒用户程序.此时,用户程序中的 MapReduce 调用返回用户代码.

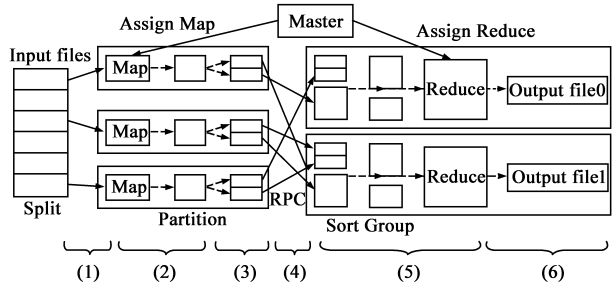

图1 MapReduce的执行过程

MapReduce 可以处理 TB 和 PB 量级的数据,能很方便地在许多机器上实现数据密集型计算的并行化.算法 1 给出基于 MapReduce 的单词统计程序伪代码,程序功能是统计文本中所有单词出现的次数.Map 函数产生每个词和这个词的出现次数(在本例中就是 1).Reduce 函数把产生的每一个特定词的计数加在一起.

算法 1　基于 MapReduce 的单词统计算法

```
Map(String key,String value):
    //key:文档的名字;value:文档的内容
    for each word w in value;
      EmitIntermediate(w,"1");
Reduce(String key,Iterator values):
    //key:一个词;values:一个计数列表
    int result = 0;
    for each v in values:
      result + = ParseInt(v);
    Emit(AsString(result));
```

3.1　编程模型改进方面的研究

Google 公司提出 MapReduce 的初衷只是为了解决与搜索相关的问题.目前,随着研究的不断深入,MapReduce 的应用范围越来越广泛,与此同时 MapReduce 的不足也越来越明显.为此,很多学者进行了相关的研究,对 MapReduce 进行了改进.

如图 1 中的步骤(5)所示,远程读取全部所需的中间值 key/value 对和按照 key 值排序会消耗掉大量的时间.为此,文献[6]提出了 Barrier-less MapReduce 并行编程模型.该模型通过修改 Reduce 函数,使得 Reduce 函数能够处理中间值 key/value 对,而不是 key/values 对来解决上述问题.然而,该模型增加了用户编程的负担,用户需要对其定义的 Reduce 函数做相应的修改才能达到上述要求.

MapReduce 缺乏支持处理多个相关异构数据集的能力.为此,文献[7]提出了 MapReduceMerge 并行编程模型,对多个异构数据集分别执行 Map 和 Reduce 操作,之后在 Merge 阶段把 Reduce 阶段已分割和分类融合的数据进行有效地合并.

MapReduceMerge 尽管能够处理多个相关异构数据集,但是并不支持自动执行多次 MapReduceMerge 过程.当用户手动执行多次 MapReduce 或者 MapReduceMerge 过程时,用户需要解决任务调度、同步性和容错等问题,编程负担过重.为此,文献[8]提出了 Oivos 并行编程模型,Oivos 利用抽象层来实现自动管理执行多次 MapReduce 或者 MapReduceMerge 过程.Oivos 需要用户指定处理多个相关异构数据集所需 MapReduce 或者 MapReduceMerge 过程的个数,并通过检测逻辑时间戳来自动发现哪些任务需要重新执行.

MapReduce 并行编程模型死板,且不能很好地适应小规模集群.为此,基于消息传递和无共享模型,文献[9]提出了名为 KPNs (Kahn process networks)的并行编程模型.KPNs 可自动执行迭代计算,且编程灵活(例如:KPNs 的输入不必抽象成 key/value 对的形式).同时,把串行算法改写成 KPNs 形式仅需在恰当的地方插入通信状态.但是该模型仍需大量的实验来验证其性能与可扩展性.

表 1　MapReduce 并行编程模型及其改进研究对比

模型名称	编程模型是否固定	能否支持迭代计算	用户使用难度	主要特点	普及程度
MapReduce	是	否	简单	Google 公司最先提出,应用广泛	普及程度较高,各大互联网公司都在使用
Barrier-less MapReduce	是	否	复杂	修改 Reduce 函数,使其能够处理中间值 key/value 对,省去排序和聚合的时间	普及程度不高,伊利诺伊大学计算机科学系独立开发
MapReduceMerge	是	否	简单	增加的 Merge 阶段把 Reduce 阶段已分割和分类融合的数据进行有效地合并	普及程度不高,雅虎公司和 UCLA 计算机科学系联合开发
Oivos	是	是	简单	自动管理执行多次 MapReduce 或者 MapReduceMerge 过程	普及程度不高,挪威特罗姆瑟大学计算机系独立开发
KPNs	否	是	简单	可自动执行迭代计算,且编程灵活,能适应小规模集群的特点	普及程度不高,挪威奥斯陆大学信息学系独立开发

表 1 显示了 MapReduce 并行编程模型及其改进研究对比.通过比较不难看出,以 Google 公司提出的 MapReduce 为基础,针对其不足,很多研究学者进行了相关研究,并取得了一定的研究成果.但是仍存在着一些不足:首先,这些模型均仅针对 MapReduce 某个方面的不足,研究较为片面,并且都没有得到广泛的应用.其次,部分模型并不成熟,仍需要进一步的实验来验证其性能与可扩展性.最后,没有研究结合不同模型的特点,得出一个全面优化的并行编程模型.

3.2　针对不同平台实现方面的研究

受到 Google 公司提出的 MapReduce 并行编程模型启发,目前被广泛使用的代表是 Hadoop.它是一个开源的可运行于大型分布式集群上的并行编程框架,提供了一个支持 MapReduce 并行编程模型的部件.Hadoop 具有良好的存储和计算可扩展性;具有分布式处理的可靠性和高效性;具有良好的经济性(运行在普通 PC 机上).

Hadoop 也存在一些不足:①小块数据处理由于系

统开销等原因处理速度并不一定比串行程序快;②如果计算产生的中间结果文件非常巨大,Reduce 过程需要通过远程过程调用来获取这些中间结果文件,会加大网络传输的开销;③Hadoop 作为一个比较新的项目,性能和稳定性的提升还需一定时间.

随着科技的进步与硬件平台的发展,许多研究学者在不同的实验平台上实现了 MapReduce,并取得了一定的研究成果.其主要的研究成果包括:

(1)在共享内存平台上:斯坦福大学计算机系统实验室在共享内存系统上实现了 MapReduce,被称为 Phoenix.Phoenix 采用线程来实现 Map 和 Reduce 任务,同时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.但 Phoenix 也存在不能自动执行迭代计算、没有高效的错误发现机制等不足.

(2)在 GPUs 平台上: Mars 是基于 GPUs(图形处理器)的 MapReduce 实现.Mars 也具有 Map 与 Reduce 这两个步骤.在每个步骤开始之前,Mars 对线程配置进行初始化 (包括:线程组的个数、每个线程组中线程的个数等).由于 GPU 线程不支持运行时动态调度,所以给每个 GPU 线程分配的任务是固定的.若输入数据划分不均匀,必将导致 Map 或者 Reduce 阶段的负载不均衡,使得整个系统性能急剧降低.同时由于 GPU 不支持运行时在设备内存中分配空间,需要预先在设备内存中分配好输入数据和输出数据的存放空间.但是在 Map 和 Reduce 阶段输出数据大小是未知的,并且当多个 GPU 线程同时向共享输出区域中写数据时,易造成写冲突.另外,由于 Mars 的编程接口是为图形处理而专门设计的,因此用户编程较为复杂.

(3)在 Cell/B.E.平台上: Cell MapReduce 是基于 Cell/B.E.(Cell 宽带引擎)的 MapReduce 实现.将 MapReduce 移植到 Cell/B.E.架构主要存在三个问题:①必须实现全局内存与 SPE 内存之间的内存管理和交换;②由于主要负责 Map 和 Reduce 任务执行的 SPE 内存是由软件管理的,因此需要通过内存交换来解决计算重叠问题;③在 Map 和 Reduce 阶段之间存在一个逻辑聚合阶段,该阶段需要在各 SPE 之间高效执行.为此,Cell MapReduce 提出:①通过块 DMA 传输来预先分配 Map 和 Reduce 任务的输出区域,以达到解决内存管理的目的;②提出通过双缓冲区和流数据的内存交换来实现计算并行化;③提出通过双阶段 Partition 和 Sort 处理来实现逻辑聚合.综上,该模型为用户提供了一个简单的机器抽象,隐藏了并行实现与硬件的细节.而且还提供了一套 API 和运行时系统来自动管理同步性、调度、分块和内存交换等工作.然而,虽然 Cell/B.E.处理单元之间有很高的带宽,但是 Cell/B.E.没有为协同处理单元提供一致性的存储器,同时程序员必须仔细管理进出于各个 SPE 的数据移动,编程难度太大.

(4)在 FPGA 平台上:近年来,FPGA(现场可编程门阵列)开始应用于高性能计算应用中[40].相对于其它并行计算平台,FPGA 具有可重构性、高灵活性和严格遵守摩尔定律的优势.但是,基于 FPGA 的计算亦受到硬件结构的设计开发和寄存器级数据传输的限制,导致编程效率较低.为此,文献[34]提出了名为 FPMR 的基于 FPGA 的 MapReduce 实现.首先,利用 FPGA 的可重构性,在片上实现多个 Map 与 Reduce 任务,以实现较好的性能.其次,通过片上动态调度策略来实现较好的资源利用率和负载均衡,以达到把编程人员从具体任务控制和通信操作中解放出来的目的.最后,通过高效的数据获取策略来实现最大化数据的重利用和解决带宽瓶颈.但是 FPMR 不支持利用页式硬件进行动态内存管理,同时也需要更多的实验来验证 FPMR 的效率和生产力.

(5)在分布式移动平台上:功能强大的移动设备的不断普及,便于提供一个功能强大的分布式移动计算环境.但是,在这样的环境中,软件开发和应用部署都面临着易出错、同步性困难、资源分配复杂、缺少编程模型支持等问题.同时,若把 MapReduce 应用于移动网络将具有很差的计算连通性.为此,文献[15]提出了名为 Misco 的基于分布式移动平台的 MapReduce 实现.Misco 由一个 Master Server 和一系列 Worker Nodes 组成.其中,Master 和 Worker 之间采用基于轮询[41]的通信机制,使用 HTTP 的方式来传输数据.Master Server 时刻跟踪用户应用,负责其任务调度与分配,保存与应用相关的输入数据、中间值和最终结果.Worker Nodes 则负责执行 Map 与 Reduce 任务.但是由于轮询的时间间隔不好确定,若时间间隔设置不当,会显著降低程序的执行性能.

(6)在公共资源网格平台上:文献[16]提出名为 Ussop 的基于公共资源网格环境的 MapReduce 实现.针对网格资源的异变性和广域网中进行数据交换开销大的特点,Ussop 提出了两个任务调度算法,一个能根据网格节点的计算能力,自适应 Map 输入的粒度.另一个能最小化在广域网传输中间数据的开销.但是 Ussop 缺乏高效的容错机制,当一个节点因处于过载状态而无法给 Map 任务分配资源时,只能在其它节点重新执行该任务,无法做到资源的重新分配或者任务迁移.

表 2 显示了 MapReduce 针对不同平台的实现研究对比.由表 2 可知,除了 Hadoop 之外,其它的实现都没有得到广泛的应用.同时一些实现(例如:Mars 和 Cell MapReduce)由于底层硬件比较复杂或者底层硬件架构的限制,造成用户编程难度较大,增加了用户负担,不利于其大范围推广.

表 2　MapReduce 针对不同平台的实现研究对比

名称	实现平台	用户使用难度	模型主要优点	模型主要缺点	普及程度
Hadoop	大型分布式集群	简单	提供一个支持 MapReduce 并行编程模型的部件;应用广泛	对小规模数据处理速度较低;若中间结果文件过大,会加大通信开销;性能和稳定性还需要提高	普及程度较高,各大互联网公司和研究机构都在使用
Phoenix	共享内存	简单	用共享内存缓冲区来通信,避免数据拷贝产生的开销	缺乏高效的错误发现机制,不支持迭代计算	普及程度不高,斯坦福大学计算机系统实验室开发
Mars	GPUs	复杂	利用众多的 GPU 线程来完成 Map 和 Reduce 的工作	若输入数据划分不均匀,易出现负载不均衡,且易发生写冲突	普及程度不高,香港科技大学与微软、新浪合作开发
Cell MapReduce	Cell/B.E.	复杂	通过块 DMA 传输来预先分配 Map 和 Reduce 任务的输出区域;提出通过双缓冲区和流数据的内存交换来实现计算并行化;通过双阶段 Partition 和 Sort 处理来实现逻辑聚合	Cell/B.E.没有为众多 SPE 提供一致性的存储器,编程难度大	普及程度不高,威斯康辛大学麦迪逊分校计算机科学系垂直研究小组开发
FPMR	FPGA	复杂	在片上实现多个 Map 任务和 Reduce 任务;利用动态调度策略来实现较高的资源利用率和负载均衡;利用高效的数据获取策略来实现最大化数据的重利用和解决带宽瓶颈	不支持动态内存管理;需要进一步验证其效率和生产力;模型不够成熟	普及程度不高,清华大学和微软亚洲研究中心合作开发
Misco	分布式移动平台	简单	Master 和 Worker 之间采用基于轮询的通信机制;使用 HTTP 的方式来传输数据	轮询的时间间隔不好确定	普及程度不高,加利福尼亚大学、雅典大学和 Nokia 研究中心合作开发
Ussop	公共资源网格平台	简单	能根据网格节点的计算能力,自适应 Map 输入的粒度;能最小化在广域网传输中间数据的开销	缺乏高效的容错策略	普及程度不高,台湾国立成功大学和立德大学合作开发

4　运行时支持库及其改进

MapReduce 运行时支持库是 MapReduce 实现的基础,它能有效进行任务调度、负载均衡、容错和一致性管理等,能够隐藏底层细节,降低用户编程的难度.

4.1　任务调度及负载均衡方面的研究

MapReduce 并行编程模型采用基于数据存储位置的任务窃取调度策略.如果某个 Worker 的本地任务列表非空,则首先执行该列表的第一个任务;如果某个 Worker 的本地任务列表为空,则从与之最近的 Worker 窃取任务来执行.这样能够减少通信开销,提高系统性能.但是,该策略也存在如下几点不足:

(1)如果数据分布不均匀,会有更多的 Map 任务不能在本地执行.此时,该策略反而会增大通信开销,导致性能下降.为此,文献[35]提出了基于已知数据分布的 MapReduce 任务调度策略.该调度策略基于节点和任务的优先级,充分考虑系统中数据的分布,将任务调度给合适的节点.这样就能实现以较高的概率将 Map 任务分配给本地有数据的节点,从而减少通信开销,提高系统性能.

(2)MapReduce 任务调度策略存在调度的公平性和数据存储位置(即为具有输入数据的节点分配任务)之间的冲突.即依照严格的任务队列顺序,一个没有本地数据的任务会被强制调度,这会增加通信开销.为此,文献[17]提出了延迟调度算法.当节点请求任务时,如果任务列表中的第一个任务不能在本地启动,则系统自动跳过该任务,查询下一个任务,直至找到一个能在本地启动的任务.当然,如果一个节点请求任务时跳过的任务过多,系统将允许其启动数据不在本地的任务.

(3)有时候,多个 MapReduce 任务需要共享相同的物理资源.这就有必要预测和管理每个任务的性能,据此为每个任务分配合适的资源.为此,文献[19]提出了基于性能驱动的任务调度策略.该策略能动态预测当前 MapReduce 任务的性能,并且能够为该任务调整资源分配,使之既能达到应用的性能目标,又不会占用过多的资源.

(4)MapReduce 为节约响应时间采用预测执行机制,即若一个节点可获得但是性能很差,那么 MapReduce 会把运行在该节点上任务作为备份任务在其它节点上运行.但是这种机制在异构环境中会导致程序性能的降低,为此,文献[18]提出了名为 LATE(Longest Approximate Time to End)的任务调度策略.LATE 通过计算所有任务的剩余时间来找到执行最慢的任务作为备份任务,这样就能有效缩短 MapReduce 在异构环境中的响应时间.但是 LATE 并不能正确计算任务的剩余时间,因而不能找到真正执行最慢的任务,同时也不能适应异构环境的动态变化.为此,文献[36]提出了名为 SAMR(Self-Adaptive MapReduce)的任务调度策略,通过记录在每个节点上的任务执行历史信息,SAMR 能动态找到真正执行最慢的任务作为备份任务.但是 SAMR 在执行备份任务时,并没有考虑数据的存储位置,同时 SAMR 仍需在不同的平台下进行评估测试.

综上,针对 MapReduce 并行编程模型采用的任务调度策略的特点,很多学者进行了相关研究,并取得了一定的研究成果.但是这些研究成果没有充分考虑数据存储位置及其动态变化,不能根据数据划分的粒度,自动选择合适的任务调度策略.

4.2 容错方面的研究

MapReduce 并行编程模型被设计用于使用数目众多的机器处理海量数据,因此 MapReduce 运行时支持库必须能够很好地处理发生的机器故障.

在 MapReduce 并行编程模型中,Worker 节点采用基于响应的错误恢复机制.Master 节点周期性地 ping 每个 Worker 节点.如果在一个时间段内 Worker 节点没有返回信息,Master 节点就会标注该 Worker 节点失效.由该失效 Worker 完成的所有任务将被重新设置成初始空闲状态,并被分配给其它 Worker 执行.

Master 节点采用基于检查点的错误恢复机制.Master 节点周期性地写入检查点.如果 Master 任务失效了,则可从最后一个检查点来启动另一个 Master 进程.

检查点和日志[42]是两种被广泛使用的错误回滚恢复机制.在 MapReduce 并行编程模型中,Worker 节点采用基于响应的错误恢复机制,Master 节点采用基于检查点的错误恢复机制.但是,基于响应的错误恢复机制不能实现低开销和高效性,基于检查点的错误恢复机制在创建检查点时开销较大.同时基于日志的错误恢复机制虽然较简单(错误恢复时所需的所有信息都在日志中),但是在跨网络传输较大的日志文件时,会造成较大的网络带宽浪费,并可能延迟应用数据的传输.为此,文献[37]提出基于日志的双阶段错误恢复机制,把日志分为 basic 和 extra 两部分,同时把恢复机制分成两阶段,以减少状态信息的传输,该机制在节省网络带宽的同时能够优化全局性能.

在 MapReduce 并行编程模型中,如果 Worker 节点发生错误,需要重启该节点上运行的所有 Map 或 Reduce 任务,不能实现低开销和高效性.为此,文献[20]提出了基于选择性的 API 来实现数据密集型应用的并行化.该 API 提供了一个由用户声明的 reduce 对象,该对象是任何节点的计算状态集合.当某个节点出错时,该节点未处理的数据可被其它节点处理.从出错节点上拷贝过来的 reduce 对象与其它节点上的 reduce 对象一起来产生最终的正确结果.这样就能实现低开销和高效的容错机制.

综上,针对 MapReduce 并行编程模型,Google 公司提出的容错策略不够完善,Master 节点容错开销过大,Worker 节点容错很容易产生重复计算,造成计算资源的浪费.对此许多研究学者进行了相关研究,并取得了一定的研究成果.但这些研究主要是针对 Worker 节点的容错,针对 Master 节点容错的研究较少.

5 总结及未来的发展趋势

目前,国内外众多研究人员已对 MapReduce 并行编程模型所涉及的关键技术(包括:模型改进、模型针对不同平台的实现、任务调度、负载均衡和容错)进行了卓有成效的研究.预计在今后的一段时期内,与 MapReduce 并行编程模型相关的研究可能会朝着以下几方面进行:

(1)逐步形成完善的 MapReduce 并行编程模型规范.它统一定义 MapReduce 并行编程模型的各个组成部分(例如:Map、Partition、Sort、Reduce 和 Merge 等),将现存的多种定义一致起来,形成能够长期有效的统一定义规范.它支持并行计算和分布式应用,具有良好的自适应能力与性能预测能力,能够满足较高的性能要求.

(2)由于 MapReduce 并行编程模型主要用于大规模数据集(TB 甚至 PB 级)的并行处理.因此,性能问题将成为研究的重点之一.可着眼于性能,研究 MapReduce 并行编程模型,在 MapReduce 并行编程模型的实现中采取多种提高性能的手段(例如:减少数据拷贝,改进节点间数据传输方法,改进运行时支持库的任务调度机制,改进内存管理,采用高效的同步机制和性能预测等).

(3)随着云计算的兴起与进一步发展,MapReduce 并行编程模型的大规模底层基础设施建设(例如:Amazon 的 EC2 与 S3[43]等)将成为研究的热点,这也是基于 MapReduce 并行编程模型的各种应用(例如:云计算等)的实现根本.

(4)针对不同的实验平台实现 MapReduce 并行编程模型.已有学者将 MapReduce 并行编程模型从最初的普通多核分布式系统,移植到共享内存和 Cell/B.E.架构等环境中.将来 MapReduce 并行编程模型会被移植到更多的实验平台上(例如:物联网[44-45]等).同时已有平台上的实现会进一步优化.

(5)MapReduce 并行编程模型的应用领域将进一步扩大.目前 MapReduce 已被各大互联网公司所采用,并且在云计算和图像处理等领域得到了广泛的应用.相信将来更多的公司会采用 MapReduce 并行编程模型,同时针对不同的应用领域,开发出更多的专用模型.

综上所述,MapReduce 并行编程模型的研究是一个充满前途和挑战的领域,它改变着大规模数据集的并行计算方式,必将在并行计算领域发挥越来越重要的作用.

参考文献

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

[2] J L Wagener. High performance fortran[J]. Computer Standards & Interfaces, Elsevier, 1996, 18(4): 371 – 377.

[3] W Gropp, E Lusk, et al. Using MPI: Portable Parallel Programming with the Message Passing Interface[M]. Cambridge: MIT Press, 1999. 1 – 350.

[4] A Geist, A Beguelin, et al. PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing [M]. Cambridge: MIT Press, 1995. 1 – 299.

[5] 廖名学,范植华. MPI 程序同步通信基本模型死锁检测[J]. 电子学报, 2008, 36(2): 402 – 407.
Liao Ming-xue, Fan Zhi-hua. Deadlock detection in basic models of MPI synchronization communication programs[J]. Acta Electronica Sinica, 2008, 36(2): 402 – 407. (in Chinese)

[6] A Verma, N Zea, et al. Breaking the mapreduce stage barrier [A]. Proc of IEEE International Conference on Cluster Computing[C]. Los Alamitos: IEEE Computer Society, 2010. 235 – 244.

[7] H C Yang, A Dasdan, et al. Map-Reduce-Merge: Simplified relational data processing[A]. Proc of ACM SIGMOD International Conference on Management of Data[C]. New York: ACM, 2007. 1029 – 1040.

[8] S V Valvag, D Johansen. Oivos: Simple and efficient distributed data processing[A]. Proc of IEEE International Conference on High Performance Computing and Communications[C]. Piscataway: IEEE, 2008. 113 – 122.

[9] Z Vrba, P Halvorsen, et al. Kahn process networks are a flexible alternative to mapreduce[A]. Proc of IEEE International Conference on High Performance Computing and Communications[C]. Piscataway: IEEE, 2009. 154 – 162.

[10] Apache hadoop [EB/OL]. http://lucene.apache.org/hadoop/, 2010-10-15/2010-12-28.

[11] C Ranger, R Raghuraman, et al. Evaluating mapreduce for multi-core and multiprocessor systems[A]. Proc of International Symposium on High Performance Computer Architecture[C]. Piscataway: IEEE, 2007. 13 – 24.

[12] R M Yoo, A Romano, et al. Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system[A]. Proc of IEEE International Symposium on Workload Characterization [C]. Piscataway: IEEE, 2009. 198 – 207.

[13] He Bingsheng, Fang Wenbin, et al. Mars: A mapreduce framework on graphics processors[A]. Proc of International Conference on Parallel Architectures and Compilation Techniques [C]. Piscataway: IEEE, 2008. 260 – 269.

[14] M d Kruijf, K Sankaralingam. MapReduce for the cell broadband engine architecture[J]. IBM Journal of Research and Development, 2009, 53(5): 747 – 758.

[15] A Dou, V Kalogeraki, et al. Misco: A mapreduce framework for mobile systems[A]. Proc of International Conference on PErvasive Technologies Related to Assistive Environments [C]. New York: ACM, 2010.

[16] Su Y L, Chen P C, et al. Variable-sized map and locality-aware reduce on public-resource grids[A]. Proc of Advances in Grid and Pervasive Computing. [C]. Berlin: Springer, 2010. 234 – 243.

[17] M Zaharia, D Borthakur, et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling[A]. Proc of EuroSys 2010 Conference[C]. New York: ACM, 2010. 265 – 278.

[18] M Zaharia, A Konwinski, et al. Improving mapreduce performance in heterogeneous environments[A]. Proc of USENIX conference on Operating systems design and implementation [C]. Berkeley: USENIX Association, 2008. 29 – 42.

[19] J Polo, D Carrera, et al. Performance-driven task co-scheduling for mapreduce environments[A]. Proc of IEEE/IFIP Network Operations and Management Symposium [C]. Piscataway: IEEE, 2010. 373 – 380.

[20] T Bicer, W Jiang, et al. Supporting fault tolerance in a data-intensive computing middleware[A]. Proc of IEEE International Symposium on Parallel & Distributed Processing[C]. Piscataway: IEEE, 2010. 1 – 12.

[21] M Kontagora, H G Velez. Benchmarking a mapreduce environment on a full virtualization platform[A]. Proc of International Conference on Complex, Intelligent and Software Intensive Systems[C]. Piscataway: IEEE, 2010. 433 – 438.

[22] K Kim, K Jeon, et al. MRBench: A benchmark for mapreduce framework[A]. Proc of International Conference on Parallel and Distributed Systems[C]. Piscataway: IEEE computer society, 2008. 11 – 18.

[23] Q Liu, T Todman, et al. Automatic optimization of mapreduce designs by geometric programming[A]. Proc of International Conference on Field-Programmable Technology[C]. Piscataway: IEEE, 2009. 215 – 222.

[24] T Sandholm, K Lai. MapReduce optimization using regulated dynamic prioritization[J]. Performance Evaluation Review, 2009, 37(1): 299 – 310.

[25] Y Becerra, V Beltran, et al. Speeding up distributed mapreduce applications using hardware accelerators[A]. Proc of International Conference on Parallel Processing [C]. Piscataway: IEEE, 2009. 42 – 49.

[26] Wei Wei, Du Juan, et al. SecureMR: A service integrity assurance framework for mapreduce[A]. Proc of Annual Computer Security Applications Conference [C]. Piscataway: IEEE, 2009. 73 – 82.

[27] Liu Qiang, T Todman, et al. Combining optimizations in automated low power design[A]. Proc of Design, Automation & Test in Europe Conference & Exhibition [C]. Piscataway: IEEE, 2010. 1791 – 1796.

[28] N Vasic, M Barisits, et al. Making cluster applications energy-

aware[A]. Proc of Workshop on Automated Control for Datacenters and Clouds[C]. New York: ACM, 2009. 37 – 42.
[29] Institutions and companies using hadoop[EB/OL]. http://wiki. apache. org/hadoop/PoweredBy, 2010-12-25/2010-12-28.
[30] 陈康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009,20(5):1337 – 1348.
Chen Kang, Zheng Wei-min. Cloud computing: System instances and current research[J]. Journal of Software, 2009, 20(5):1337 – 1348. (in Chinese)
[31] U Kang, C E Tsourakakis, et al. PEGASUS: A peta-scale graph mining system-implementation and observations[A]. Proc of IEEE International Conference on Data Mining[C]. Piscataway: IEEE, 2009. 229 – 238.
[32] Liu Yang, Jiang Xiao-hong, et al. MapReduce-based pattern finding algorithm applied in motif detection for prescription compatibility network[A]. Proc of International Symposium on Advanced Parallel Processing Technologies[C]. Berlin: Springer, 2009. 341 – 355.
[33] Yang Lai, Shi Zhong-zhi. An efficient data mining framework on hadoop using java persistence api[A]. Proc of IEEE International Conference on Computer and Information Technology [C]. Los Alamitos: IEEE Computer Society, 2010. 203 – 209.
[34] Shan Yi, Wang Bo, et al. FPMR: MapReduce framework on FPGA a case study of rankboost acceleration[A]. Proc of ACM SIGDA International Symposium on Field-Programmable Gate Arrays[C]. New York: ACM, 2010. 93 – 102.
[35] Guo Lei-tao, Sun Hong-wei, et al. A data distribution aware task scheduling strategy for mapreduce system[A]. First International Conference on Cloud Computing [C]. Berlin: Springer, 2009. 694 – 699.
[36] Chen Quan, Zhang Da-qiang, et al. SAMR: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment [A]. Proc of IEEE International Conference on Computer and Information Technology[C]. Los Alamitos: IEEE computer society, 2010. 2736 – 2743.
[37] Chen Ting, Wang Yong-jian, et al. A two-phase log-based fault recovery mechanism in master/worker based computing environment[A]. Proc of IEEE International Symposium on Parallel and Distributed Processing with Applications[C]. Piscataway: IEEE, 2009. 290 – 297.
[38] Huang Sheng-sheng, Huang Jie, et al. The hibench benchmark suite: characterization of the mapreduce-based data analysis [A]. Proc of IEEE International Conference on Data Engineering Workshops[C]. Piscataway: IEEE, 2010. 41 – 45.
[39] Zhang Shu-bin, Han Ji-zhong, et al. Accelerating mapreduce with distributed memory cache[A]. Proc of IEEE International Conference on Parallel and Distributed Systems[C]. Piscataway: IEEE, 2009. 472 – 478.
[40] M C Herbordt, T V Court, et al. Achieving high performance with FPGA-based computing[J]. Computer, 2007, 40(3): 50 – 57.
[41] K Langendoen, J Romein, et al. Integrating polling, interrupts, and thread management[A]. Proc of Symposium on the Frontiers of Massively Parallel Computation[C]. Los Alamitos: IEEE computer society, 1996. 13 – 22.
[42] M Treaster. A survey of fault-tolerance and fault-recovery techniques in parallel systems[EB/OL]. http://arxiv. org/abs/cs/0501002, 2005-01-01/2010-12-28.
[43] A Muni, J Hansen. Amazon web services[J]. Dr. Dobb's Journal, 2005, 30(12): 66 – 67.
[44] 宁焕生,徐群玉. 全球物联网发展及中国物联网建设若干思考[J]. 电子学报,2010,38(11):2590 – 2599.
Ning Huan-sheng, Xu Qun-yu. Research on global internet of things' developments and it's construction in china[J]. Acta Electronica Sinica, 2010, 38(11): 2590 – 2599. (in Chinese)
[45] 宁焕生,张瑜,等. 中国物联网信息服务系统研究[J]. 电子学报,2006,34(12A):2514 – 2517.
Ning Huan-sheng, Zhang Yu, et al. Research on China internet of things' services and management[J]. Acta Electronica Sinica, 2006, 34(12A): 2514 – 2517. (in Chinese)

作者简介

李建江 男,1971 年生于四川广安,博士,副教授,CCF 会员,主要研究方向为高性能计算、并行编译和并行软件工程.
E-mail: jianjiangli@gmail.com

崔 健 男,1986 年生于江苏苏州,硕士生,主要研究方向为高性能计算、并行软件工程.
E-mail: cuijian613@yahoo.com.cn