Research on Energy-Consumption Optimization Approaches for Big Data Processing Platform

大数据处理平台能耗优化方法的研究

宋 杰 著

大数据处理平台能耗优化方法的研究

Research on Energy-Consumption Optimization Approaches for Big Data Processing Platform

宋 杰 著

东北大学出版社
·沈 阳·

图书在版编目（CIP）数据

大数据处理平台能耗优化方法的研究 / 宋杰著. — 沈阳：东北大学出版社，2016.11

ISBN 978-7-5517-1476-1

Ⅰ. ①大… Ⅱ. ①宋… Ⅲ. ①数据处理—最优化算法 Ⅳ. ①TP274

中国版本图书馆 CIP 数据核字（2016）第 284730 号

出 版 者：东北大学出版社
地址：沈阳市和平区文化路三号巷 11 号
邮编：110819
电话：024－83687331（市场部）　83680267（社务部）
传真：024－83680180（市场部）　83687332（社务部）
网址：http://www.neupress.com
E-mail：neuph@neupress.com
印 刷 者：沈阳航空发动机研究所印刷厂
发 行 者：东北大学出版社
幅面尺寸：170mm×240mm
印　　张：12
字　　数：239 千字
出版时间：2016 年 11 月第 1 版
印刷时间：2016 年 11 月第 1 次印刷
组稿编辑：罗　鑫
责任编辑：潘佳宁　刘　泉
责任校对：罗　鑫
封面设计：刘江旸
责任出版：唐敏志

ISBN 978-7-5517-1476-1
定　　价：45.00 元

前言

大数据时代的到来将使处理数据所用的计算机数量快速增长，日益普及的大数据系统所导致的大量能耗问题已引起政府和业界的重视。大数据处理平台是大数据系统中最重要的一类，但现有绿色计算的研究成果尚不能完全有效地适用于大数据处理平台，尚缺少有针对性的基于中间件的能耗优化方法。本书阐述了笔者三年来在该领域所取得的研究成果，重点围绕大数据处理平台中“缩减节点的等待时间以减少空闲能耗”这一能耗优化方法。首先综述了大数据处理平台和能耗优化研究进展；随后从评价模型和核心技术两个层面展开论述。模型层面介绍如何建立大数据处理平台的能耗模型和求解最小能耗值，以及主流 NoSQL 数据库系统的能耗基准测试方法和测试结果。技术层面介绍：通过优化资源分配和任务调度策略，消除资源瓶颈，减少资源等待；或通过优化数据布局策略，提高节点并行性，减少节点间等待；最终降低平台能耗。本书最后探讨了大数据处理平台能耗优化的进一步思路，及面向大数据处理算法的能耗优化。

通过阅读本书，读者能够更加深入理解大数据处理平台的核心技术和能耗优化的学术前沿，帮助读者更加有效地构建平台，或对现有平台进行改进，以及开展大数据和绿色计算领域的交叉研究。本书所述内容有利于研发大数据处理平台，推动大数据技术的应用，具有重要的科学意义和广泛的应用前景。

作　者

2016 年 7 月

目录

第1章 绪 论

当提及节能减排技术时，人们会想到制造业、交通运输等传统行业，殊不知，计算机等IT设备的电能消耗同样不容忽视。各种数据表明，计算机正在吞噬大量能源，在不知不觉中给环境带来沉重压力。2008年，全球接入Internet的IT资源的年耗电量等同于14个大型发电站的年供电量，换算成二氧化碳排放量，相当于全球航空公司一年的碳排放量之和[1]。目前大量的运算资源用于提供超大规模计算或数据管理服务，而且规模在迅速扩大。2000年至2005年，全球的服务器数量增长了1倍多。2005年11月的统计数据表明，世界上最强大的500个超级计算机系统共拥有73.2万个处理器，而2010年11月则增加为647.2万个，五年间增长了近百倍[2]；以天河一号(TH-1)超级计算机系统为例，其包含6144个通用处理器和5120个加速处理器，其内存高达229376 GB[3]。TH-1的功率为4040kW，而Jaguar超算系统功率则高达6950kW[2]。即使不考虑TH-1的制冷等辅助设备的耗电量，其电能耗等同于3万个家庭的生活用电，维护这样一个超算系统的电力资源费用是惊人的。

大数据时代的到来并不会减少IT资源，相反，存储和处理大数据需要大量的计算机和集群系统。计算机能耗在未来十年还会快速地增长。一方面，硬件的价格在逐年降低，人们可以花费更少的经费而购得更强大的服务器；而另一方面，对集群系统这种大规模计算模式的需求也将会迅速增加，需要更多硬件的投入。表面上看，这是一个良性的发展，然而，潜在的问题是能源价格的逐年升高，廉价的基础设施却会带来昂贵的能源开销[4]。如果硬件能耗费用超过了硬件价格本身，将极大阻碍IT行业的发展。摩尔定律引发了计算机以高运算速率和高能耗为导向的发展，但未必符合能源和环境成本，硬件成本的下降和能耗的降低不成正比。可以预测，高能效计算(energy efficient computing)将成为下十年最为迫切且最具有挑战性的课题之一。

大数据处理平台多采用云计算和集群计算技术，尽管云计算系统被认为是一种绿色计算系统，但其本身并没有提供成熟的解决方案来评价和降低能耗，仍需要一系列能耗优化方法来切实地实现“绿色计算”。近年来研究的绿色数据中心(green data center)、高能效分布式系统(energy efficient large-scale distributed systems)等研究尚不能完全有效地适用于大数据处理平台。原因如下。

首先，通过中间件进行资源分配、任务调度来实现硬件系统的节能，其节能效果要优于单纯地使用硬件和操作系统级别的节能技术，这种优势在集群系

统中尤为明显，因为中间件节能方法更针对具体应用特征(如负载特征)，可控粒度更大(如可动态关闭计算机节点)[4-6]。大数据处理平台属于一种特点鲜明的集群系统，但目前恰恰缺少针对大数据处理平台的中间件节能方法。

其次，主流的节能思路有两种，一是静态节能，即研究更低功耗的硬件设备；二是动态节能，即动态地关闭空闲“组件”以减少空闲能耗。对于后者，硬件节能方法中的“组件”多为芯片；操作系统节能方法中的“组件”多为计算机部件；面向集群系统的中间件节能方法中“组件”多为计算机节点。然而，实验结果表明，对于大数据处理平台，每个节点不仅要参与运算，还更多地参与数据管理和数据提供，从数据的可用性、完整性和一致性角度，节点空闲的时机很少，动态开关节点的代价很大。例如，大数据处理平台中数据是多副本的，首先要保证至少有一份数据在线，且若某节点关闭一段时间后再次开启，它需要和其他节点的数据副本进行同步。由此可见，现有研究尚不能良好地优化大数据处理平台能耗，如不能找到一种适用于大数据处理平台的能耗优化方法，则绿色计算只能成为一种理想。

本书阐述大数据处理平台中“缩减节点的等待时间以减少空闲能耗”这一能耗优化方法，主要从评价模型和核心技术两个层面展开介绍。在综述大数据处理平台和能耗优化研究成果之后，本书首先介绍能耗评价模型的建立和优化求解，以及主流大数据处理平台的能耗基准测试方法和测试结果；随后介绍能耗优化的资源分配和任务调度策略、数据布局策略、作业执行策略，减少节点因等待而造成的“被动”空闲，降低能耗。

首先，迫切需要一个模型来评价大数据处理平台的能耗。例如，在评价汽车的油耗时，通常使用百公里耗油量(升)作为油耗的度量，而且辅以很多约束，如行驶环境、客载量、计算方法等，因此可以认为百公里油耗评价模型是一个直观且准确的汽车油耗度量，同样需要这样的模型来评价大数据处理平台的电能耗。此外，评价大数据处理平台能耗，还需要定义一系列能耗基准测试用例。由于大数据处理平台的特殊性和不成熟性，经典 TPC-H 的诸多查询过于复杂而难以支持(如目前 HBase 和 Cassandra 尚不提供直观的表连接和分组聚集操作接口)，因此需要定义一组新的能耗基准测试用例，包括用例描述、数据集结构、数据生成方法、测试环境(数据量、节点个数、并发访问个数等)和能耗测量方法的定义。综上，有了能耗评价模型，可以更准确地评价大数据处理平台的能耗，评价能耗优化的效果，更全面地分析能耗规律，并静态地通过调整相关属性来优化能耗。

其次，在负载饱满的情况下，大数据处理平台的节点很少真正意义的空闲，而更多的是节点(或节点组件)因等待其他资源而处于“被动空闲”，这种空闲既影响性能，又浪费电能，且难以通过“动态开关节点”优化。定义在集群环境中，计算机节点(或节点组件)因等待其他资源而处于被动空闲的

“等待时间”内消耗的能量称为“等待能耗”。例如，当集群中某个节点等待其他节点的运算结果时，节点产生等待能耗，该节点并不是真正意义上空闲，通常无法关闭该节点以节能。同理，当任务在节点上执行时，若给任务分配的资源不合理，各个资源之间也会产生等待，如CPU会因等待I/O操作而阻塞，产生等待能耗。因此应该尽量减少节点等待时间和等待能耗。

针对大数据处理平台特点，本书论述“缩减节点的等待时间以减少空闲能耗”这一能耗优化方法。前期试验证明导致节点空闲的原因有三：其一，任务占有的各个资源比例不合理，任务执行存在瓶颈，资源间相互等待(参考任务隔离的概念，如虚拟机和任务槽)；其二，节点间因任务执行不同步而相互等待；其三，节点等待网络数据的传输。定义用户对大数据处理平台的请求(request)可以转换为多个作业(job)，而作业由若干任务(task)来完成，各任务(的实例)在多个节点上并行执行(同Hadoop MapReduce中作业和任务的定义)。对于原因一，通过能耗优化的资源分配和任务调度，让每个节点充分利用资源，每个任务获得各资源的比例合理，减少瓶颈产生；对于原因二，由于大部分大数据处理平台采用“迁移计算到数据端”的MapReduce模型，因此本书拟通过调整数据布局，提高各个节点的并行性；对于原因三，本书将从大数据处理算法角度加以论述。综上，本书针对不同任务种类，采用合理的资源分配策略、数据布局策略和作业执行策略，减少节点对资源、其他节点及网络数据传输的等待，缩减节点等待能耗，进而优化能耗。本书整体思路如图1.1所示：

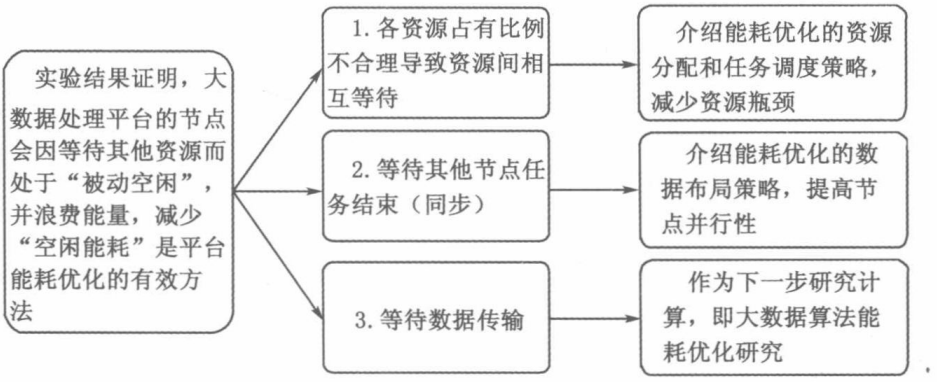

图1.1 大数据处理平台能耗优化的思路

基于现有高能效计算的研究成果，依据大数据处理平台的特点和前期的研究成果，制订本书主要内容为：大数据处理平台和能耗优化综述；模型层面阐述能耗评价模型和基准测试；技术层面阐述能耗优化的任务调度策略、数据布局策略；以及下一步研究展望。详细内容如下：

第1章：绪论；

第2章：综述主流的大数据处理平台；

第3章：综述主流的大数据处理平台能耗优化方法；

第4章：阐述大数据处理平台的能效度量模型，分析平台导致平台低能效的原因，论述优化思路；

第5章：介绍NoSQL数据库系统能耗的基准测试方法，对主流系统进行能耗基准测试，分析测试结果和系统高能耗的原因，论述优化思路；

第6章：介绍能效优化的MapReduce资源比模型，以及按最佳资源的资源分配思路；

第7章：介绍优化MapReduce系统能耗的任务分发算法；

第8章：介绍优化MapReduce系统能耗的数据布局算法；

第9章：综述了主流的大数据处理算法，提出面向处理算法的能耗优化新思路。

本书所述内容有着较强的科学意义和应用前景。首先，大数据处理平台处于起步阶段，其定义、协议、策略及实现都处于未成熟的阶段，其能耗研究的机遇和挑战并存。大部分科研都集中在“大数据处理平台实现”，“大数据处理算法的适用性和性能优化”，以及“大型分布式系统或Internet数据中心的物理层能耗介绍”上，本书所述内容具有一定的前沿性。其次，第2章将详述，大数据处理平台是近期最热门的数据处理平台，各种大数据处理平台雨后春笋般涌现。可以预测，大数据处理平台在未来十年将迅速普及。这些都是本书所述内容良好的应用环境。最后，本书所述方法潜在的生态和经济影响极为明显，目前国内电能主要是以火力发电为主，能耗的降低不仅节省了开销，也减少了二氧化碳的排放量，保护了环境，符合目前倡导的节约型经济和低碳经济。综上所述，本书内容具有先导性、前沿性和实用性。

第 2 章　大数据处理平台

近年来，伴随着信息技术、互联网和物联网技术的不断发展，数据采集终端迅猛增加，人们步入信息爆炸的大数据时代。正如麦肯锡所说：“数据，已经渗透到当今每一个行业和业务职能领域，成为重要的生产因素。人们对于海量数据的挖掘和运用，预示着新一波生产率增长和消费者盈余浪潮的到来”。在大数据时代，商业、经济及其他领域中的决策将不再基于经验和直觉，而是基于大数据分析结果，因此大数据分析处理技术已经成为一个重要的研究和应用领域。同时，业界对于该技术的急切需求，以及集群计算技术的成熟，促使各种基于大规模分布式系统的大数据处理平台和处理算法如雨后春笋般涌现[7]。大数据处理采用分治法，将大数据问题分解成规模较小的子问题求解，然后合并子问题的解，从而得到最终解。基于此，Google 公司研发的 MapReduce 是一种专门处理大数据的编程模型和实现框架，具有简单、高效、易伸缩及高容错性等特点[8]。

Google MapReduce 在设计之初致力于通过大规模廉价服务器集群实现大数据的并行处理，它优先考虑系统的伸缩性和可用性，用于处理互联网中海量网页内容数据，通过存储、索引、分析及可视化等处理步骤，实现用户对网页内容的搜索和访问[9]。MapReduce 在一个简单的库中隐藏分布式执行、容错、数据分发、任务调度及负载均衡等难点，隐藏远程数据访问、节点失效和任务间通信等细节。而 MapReduce 之所以能够迅速成为大数据处理的主流计算平台，得力于其自动并行、自然伸缩、实现简单和支持商用硬件等特性[10]。现如今，MapReduce 已是成熟的 TB/PB 级大数据处理平台，广泛地应用于社交网络、科学数据分析、传感器数据处理、医疗和电子商务应用中，并拥有各种不同版本的实现。本章分析和对比典型 MapReduce 大数据处理平台，介绍其优劣势及适用范围。

大数据处理平台是一种“计算平台”，计算平台泛指支持算法执行的硬件系统、操作系统和运行库[11]，那么大数据处理平台则泛指可以支持大数据处理算法执行的平台。大数据处理平台采用集群系统作为硬件环境，分布式中间件作为数据存储和计算平台；采用无共享体系结构，数据处理程序部署在每个节点之上；数据保存在分布式文件系统中。本章将综述基于 MapReduce 的大数据处理平台，分析它们的性能特征。

2.1 平台描述

由于 MapReduce 具有简单、易伸缩性及高容错的特点，对大数据具有高效批处理能力，Yahoo、Facebook、Amazon 和 IBM 都将 MapReduce 作为大数据处理平台。Apache 研发的 Hadoop MapReduce 最为流行，并以此引出完善的 Hadoop 生态圈。作为 Hadoop MapReduce 的良好补充，业界和学界针对不同设计目标，实现了多种 MapReduce 平台。本章将分析对比典型的 MapReduce 大数据处理平台，如 Hadoop[12]，GridGain[13]，Mars[14]，Phoenix[15]，Disco[16]，Twister [17]，Haloop[18]，iMapReduce[19]，iHadoop[20] 和 PrIter[21]，以及类似 MapReduce 的 Dryad[22]，Spark[23]。

2.1.1 Hadoop

Hadoop 能很好地支持 Java 语言编写的 MapReduce 作业，通过 Hadoop Streaming 或 Hadoop Pipes 工具也能支持 C/C + + 或其他语。Hadoop 为大规模并行数据处理算法提供运行环境，其工作原理为：将作业分解成更小的任务，将数据进行分区，每一个任务实例处理一个不同的分区，任务实例并行执行[24]，这充分体现了分治的思想。

Hadoop 把 MapReduce 作业分解成顺序执行的 Map 阶段和 Reduce 阶段，Map/Reduce 阶段包含一个 Map/Reduce 任务，Map/Reduce 任务的实例(简称实例)部署到 Map/Reduce 节点并行执行，当所有 Map/Reduce 实例执行结束后，Map/Reduce 阶段才结束。一个 MapReduce 程序仅包含两个函数，即 Map 函数和 Reduce 函数，它们定义了用户处理“键值对数据”的 Map 任务和 Reduce 任务。程序的输入数据集位于分布式文件系统中，采用“迁移运算而非迁移数据”的方式，Map/Reduce 任务被下载到每个数据节点并执行，输出结果仍保存在分布式文件系统中[25]。

Map 和 Reduce 函数的输入和输出都是用户定义格式的键值对形式。Map 函数输入 $<Key_M^{In}, Value_M^{In}>$，输出 $<Key_M^{Out}, Value_M^{Out}>$，Reduce 函数的输入为 $<Key_R^{In}, Value_R^{In}>$，输出为 $<Key_R^{Out}, Value_R^{Out}>$，其中 Key_M^{Out} 需要能隐式地转换为 Key_R^{In}，$Value_R^{In}$ 是保存 $Value_M^{Out}$ 的集合。

MapReduce 将作业(job)分解为任务(task)并在每个节点上并行地执行。MapReduce 程序首先将输入数据分割成 M 份(通常 M 大于节点个数)，作为 M 个 Map 实例的输入。Map 函数从一份输入中读取每一条记录，对其进行必要

的过滤和转换，然后输出 $<\text{Key}_{\text{M}}^{\text{Out}}, \text{Value}_{\text{M}}^{\text{Out}}>$ 格式的中间结果。这些中间结果被 Hash 函数按 $\text{Key}_{\text{M}}^{\text{Out}}$ 分割为 R 个不相交的组，每个组被写入到处理节点的本地磁盘中。所有 Map 函数终止时，M 个 Map 实例把 M 份输入文件映射成 $M \times R$ 个中间文件。由于所有 Map 函数的分割函数都一样，因此相同键 Hash 值（设为 j）的 Map 函数输出结果被存在文件 $F_{ij}(1 \leqslant i \leqslant M,\ 1 \leqslant j \leqslant R)$ 中。

MapReduce 的第二阶段执行 R 个 Reduce 实例，R 通常是节点的数量。每个 Reduce 实例 R_j 输入文件为 $F_{ij}(1 \leqslant i \leqslant M)$。这些文件从各个节点通过网络传输汇聚到执行节点。Reduce 函数输入 F_{ij} 的键 $\text{Key}_{\text{M}}^{\text{Out}}$ 和对应的一组 $\text{Value}_{\text{M}}^{\text{Out}}$ 值，并向分布式文件系统输出若干 $<\text{Key}_{\text{R}}^{\text{Out}}, \text{Value}_{\text{R}}^{\text{Out}}>$ 格式的记录。所有的从 Map 阶段产生的具有相同 Hash 值的 $<\text{Key}_{\text{M}}^{\text{Out}}, \text{Value}_{\text{M}}^{\text{Out}}>$ 输出条目均被相同的 Reduce 实例处理。

输入数据集以集合的形式存在于分布式文件系统中的一个或多个分区中。MapReduce 的调度器决定执行多少个 Map 实例，以及如何把它们分配给可用的节点；同样，调度器还必须决定运行 Reduce 实例的数量和执行位置（Hadoop 早期版本让用户指定 Map 和 Reduce 实例的数目）。MapReduce 中央控制器协调每个节点上的运算，一旦最终结果以新数据的形式写入分布式文件系统中，MapRedcue 作业执行完毕。MapReduce 作业的执行过程如图 2.1 所示。

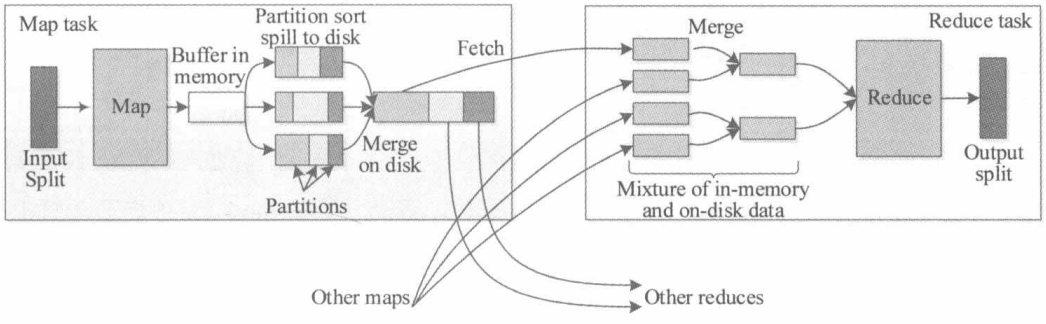

图 2.1　Hadoop MapReduce 作业执行流程

2.1.2　GridGain

GridGain 是另一种 MapReduce 的开源实现，GridGain 仅支持 Java 语言，它与 Hadoop DFS 相兼容。GridGain 提供了一种分布的、基于内存的、实时的和易伸缩的数据网格，将数据源和各类数据处理程序连接起来。GridGain 与 Hadoop 相比较最大的差异在于，前者是一个内存版本的 MapReduce，而后者能够支持超大规模数据集的处理。两者的具体差异在于：① GridGain 的作业仅包含单一的 Reduce 实例，因此在 Reduce 阶段 GridGain 不具备并行性；② GridGain 的 Map 任务仅返回单一值，若要返回多个值，则需要将多个值封装为

集合；③ GridGain 对 Map 任务输出的中间结果不排序，也不会合并；④ GridGain 的作业管理策略灵活，用户可以随意创建和终止作业，作业的输入和输出格式也较 Hadoop 更为自由；⑤ GridGain 的 Map 任务强制在节点本地执行，本地性强，任务执行效率高，网络 I/O 的开销小，但容易导致“木桶效应”，在同步时执行快的节点会等待执行慢的节点。综上所述，GridGain 和 Hadoop 在 MapReduce 模型上并无本质差异，两者不同之处主要在于任务调度、接口定义和附功能细节。

2.1.3 Mars

Mars 是一种基于 GPU 的 MapReduce 框架，能够在 GPU 上正确、有效、简易地执行数据密集型和计算密集型作业。Mars 简洁的编程接口向用户隐藏了 GPU 编程的复杂性，并且能够自动地实现 CPU 和 GPU 的任务分割、数据分发、并行化和线程管理。Mars 在 GPU 内启动大量的线程，再均衡地分配任务实例至每线程，每线程执行一个 Map/Reduce 实例。Mars 以小数量的键值对数据作为任务实例的输入，并通过一种无锁的方法来管理多任务实例对数据的并发写操作。

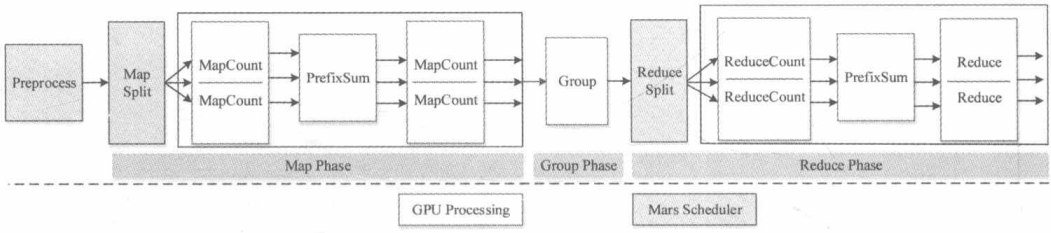

图 2.2 Mars 作业执行的各个阶段

Mars 作业执行的各个阶段如图 2.2 所示：包括 Map、Group 和 Reduce 三个阶段，每个阶段对应多个任务实例，调度器将 Map 和 Reduce 实例调度至 GPU 上运行，并将结果返回给用户。① 首先，数据存储在磁盘中，在 Map 阶段之前，Mars 会对磁盘数据进行预处理，在主存中将输入数据转换成键值对格式。② 接着，调度器初始化线程配置，定义 GPU 线程组的数量和每个线程组中线程的数量。③ 随后 Map 阶段开始，MapSplit 将输入数据由内存调度至 GPU 线程，平衡所有线程的负载，每个线程执行 MapCount 函数来计算 Map 实例输出的中间结果的数量(种类)和规模，并统计出一个局部直方图，然后在局部直方图上执行前缀和(Prefix Sum)函数，获得每个线程的输出大小及写入位置。每一个 GPU 线程执行用户自定义的 Map 函数并且输出中间结果至缓存(内存)；正因为所有线程的输出位置是经过预计算而确定的，因此不会产生写冲突，也不需要并发读写锁机制。④ 下一个阶段是 Group 阶段，该阶段可

对中间结果进行排序和 Hash 分组，也可以实现先分组再组内排序。⑤ 在 Reduce 阶段，ReduceSplit 将具有相同 Key 值的分组调度到同一 GPU 线程，同 Hadoop MapReduce 一样，这种方式也会导致负载不均衡。由此可见，Mars 和 Hadoop 对 MapReduce 作业的执行方式基本相同，但前者可以以最小的代价快速部署代码至分布式环境，并充分整合 GPU 资源。

2.1.4　Phoenix

Phoenix 的 MapReduce 实现方法和 Hadoop MapReduce 基本相同，其特点在于 Phoenix 是一种共享内存的分布式计算平台，该特点能最小化任务分发和数据通信所导致的网络 I/O 代价。Phoenix 采用纯 C + + 编写，提供 C/C + + 的应用程序接口。Phoenix 适用于多核或多处理器系统，能够对用户屏蔽多核编程、并行化、资源管理及自动容错等功能的复杂性[26]。Phoenix 通过共享内存的多线程执行 Map/Reduce 实例实现并行数据处理，其原理如图 2.3 所示。

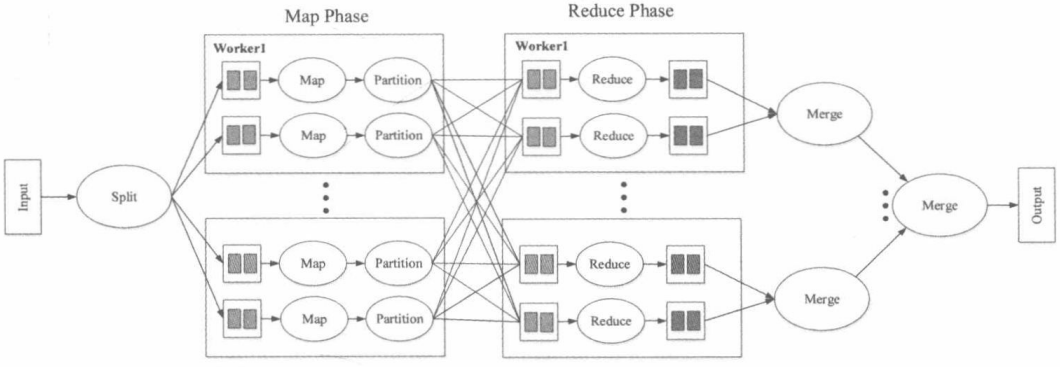

图 2.3　Phoenix 数据流

用户首先编写 Map/Reduce 函数，并指定待处理的数据，随后 Phoenix 在多个 CPU 上启动多个线程。在 Map 阶段，输入数据会被划分成多个块，在每个块上调用 Map 函数处理数据，将键值对中间结果输出至内存。在 Reduce 阶段，将相同键对应的所有值传递给 Reduce 函数，Reduce 函数将输入约减为单一的键值对，所有 Reduce 实例的输出结果最终被合并、排序和输出。

2.1.5　Disco

Disco 是 Nokia 研究中心研发的 MapReduce 轻量级开源实现，其核心组件采用 Erlang 语言开发，外部编程接口为 Python 语言。Disco 支持大数据集的并行处理，能运行在低可靠性的集群系统之上，也可以部署在多核计算机、Amazon EC2 等云平台之上。