第 42 卷 第 6A 期
2015 年 6 月

计 算 机 科 学
Computer Science

Vol. 42 No. 6A
June 2015

MapReduce 并行编程模型研究综述

杜 江 张 铮 张杰鑫 邰 铭

(解放军信息工程大学 郑州 450001) (数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 MapReduce 并行编程模型的出现简化了并行编程的复杂度。通过调用方便的接口和运行时支持库,MapReduce 并行编程模型可令大规模并行计算任务自动并发地执行而不必关心底层的具体实现细节,从而令 MapReduce 并行编程模型在大规模中低性能集群中发挥出色的计算能力,且可节约成本。对国内外关于 MapReduce 并行编程模型的研究现状进行了综述,分析了目前国内外相关研究成果的优缺点,并对 MapReduce 并行编程模型的未来发展进行了展望。

关键词 MapReduce,并行编程模型,并行计算,海量数据处理

中图法分类号 TP393 文献标识码 A

Survey of MapReduce Parallel Programming Model

DU Jiang ZHANG Zheng ZHANG Jie-xin TAI Ming

(PLA Information Engineering University, Zhengzhou 450001, China)

(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract MapReduce parallel programming model simplifies the complexity of parallel programming. Through calling a convenient interface and runtime support libraries, MapReduce parallel programming model makes large scale parallel computing tasks automatically execute concurrently without caring about the underlying implementation details, thus it can exert significant computing power in the large-scale low-performance cluster, which is cost saving as well. This paper reviewed the research of MapReduce parallel programming model at home and abroad, analysed the strengths and weaknesses of current research achievements, and prospected the future trend for the MapReduce.

Keywords MapReduce, Parallel programming model, Parallel computing, Massive data processing

1 引言

MapReduce[1] 是谷歌公司于 2004 年提出的能够对海量数据进行并行处理的编程模型,拥有简单适用的特点,被广泛应用。MapReduce 能够隐藏分布式计算的底层实现细节,降低程序员进行并行编程的难度,从而让程序员从复杂的并行编程的代码中解脱出来,轻松编写简单、高效的并行程序。

最初 Google 公司提出的 MapReduce 并不是开源的编程模型,Nutch 的开发人员把它的设计思想进行了开源实现。Nutch 是由 Doug Cutting 在 2002 年创建的一个大型全网搜索引擎项目,包括索引查询、网页抓取等功能。随着网络数据的爆炸式膨胀,Nutch 遇到了严重的扩展性问题,无法完成数十亿网页的存储和索引。而谷歌在 2003 年发表的关于谷歌分布式文件系统(GFS)的论文[2],描述了谷歌搜索引擎中网页数据的存储架构,其思路解决了 Nutch 遇到的海量网页的存储问题。由于没有公开源码,Nutch 自己完成了 GFS 的开源实现,名为 NDFS(Nutch Distributed File System)。2004 年谷歌发表的另一篇论文,描述了谷歌内部使用的分布式计算框架 MapReduce,该框架可以解决 Nutch 遇到的海量网页的索引问题。但是谷歌依然没有公开源码。而 Nutch 再一次完成了该框架的开源实现,名为 Nutch MapReduce。MapReduce 具有分布式计算能力和简单适用的特点因而适用于搜索领域,2006 年初开发人员将其移出 Nutch,成为 Lucene 的一个子项目,命名为 Hadoop。原 NDFS 也更名为 HDFS(Hadoop Distributed File System),与 MapReduce 一起组成了 Hadoop。大约同一时间,Doug Cutting 加入雅虎公司,且公司同意组织一个专门的团队继续发展 Hadoop。同年 2 月,Apache Hadoop 项目正式启动以支持 MapReduce 和 HDFS 的独立发展。2008 年 1 月,Hadoop 成为 Apache 顶级项目,迎来了它的快速发展期。

MapReduce 编程模型最大的优势在于隐藏了分布式计算的底层实现细节,用户不需要关心分布式集群的底层架构,只需要专注于 Map 和 Reduce 程序的编写,大大降低了用户的编程难度。作为分布式计算模型,集群可以由廉价的商用

本文受 863 计划重点项目:新概念高效能计算机体系结构及系统研究开发(2009AA012200),上海市科研计划项目:新概念高效能计算机体系结构及系统研究开发(08dz1501600),上海市科研计划项目:拟态安全原理验证平台研制(13dz1108800)资助。

杜 江(1990—),男,硕士生,主要研究方向为高性能计算,E-mail:kidding.1412@163.com;张 铮(1976—),男,博士,副教授,主要研究方向为高性能计算;张杰鑫(1989—),男,硕士生,主要研究方向为包分类技术、高性能计算;邰 铭(1967—),男,硕士,副教授,主要研究方向为计算机软件与理论。

机组成，大幅度节约了集群成本。MapReduce 还提供了一个运行时支持库，它可以使任务自动并行执行，提供了方便用户对任务调度策略、负载均衡、容错和一致性等机制进行高效管理的接口。

2 MapReduce 总体研究现状

最近几年，由于对海量数据（TB 和 PB 级）处理的需求，MapReduce 已经逐渐成为了最受欢迎的并行编程模型之一。作为云计算、并行计算模型的研究热点，国内外对 MapReduce 的研究文献越来越多。主要研究方向及研究成果有以下几个方面：

1. 对 MapReduce 并行编程模型的改进方面：MapReduce 存在着诸多的不足与限制，无法对每一种特定的作业都表现出卓越的性能。针对一些特定的作业应用场景，学者们对 MapReduce 的编程框架也进行着相关的改进。目前比较典型的研究成果有：Barrier-less MapReduce[3]、MapReduce-Merge[4]、Dache[5]、MARCO[6]。但是这些改进的模型均只针对 MapReduce 某方面的不足，没有得到广泛的应用。

2. 关于 MapReduce 调度器设计与优化：MapReduce 在处理海量数据的过程中，会有大量的作业被分成海量的 Map 任务和 Reduce 任务，将任务分配给合适的计算节点是调度器的职责。与 MapReduce 编程模型相似，优秀的调度器往往是针对某一特性作业池的调度或者针对某种调度需要。对具体业务有针对性的调度器很大程度上决定了 MapReduce 的使用性能。目前比较典型的研究包括：耦合调度[7]、洗牌联合调度[8]、数据与计算本地性研究[9,10]和资源感知调度[11]。

3. MapReduce 及其相关的具体应用工具方面：由于 MapReduce 在海量数据处理中的广泛使用，与之相关的应用工具也得到了人们的广泛研究和改进。其中最常用的就是 Hive。Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的 sql 查询功能，可以将 sql 语句转换为 MapReduce 任务进行运行。在 MapReduce 的具体应用工具方面的研究成果主要有 YSmart[13]、JackHare[14]和 Tarazu[15]。

4. 除了对 MapReduce 本身框架、调度、算法的研究，也有很多在具体应用背景下对 MapReduce 的应用范例，包括在机器学习背景下利用 MapReduce 进行大规模深信度网络搭建[16]、分布式数据管理[17]、生物信息学中的生物测序[18]、模拟大规模并行规约算法[19]等。

3 MapReduce 编程模型改进

3.1 MapReduce 并行编程模型

从 MapReduce 自身的命名特点可以看出，MapReduce 由两个阶段组成：Map 和 Reduce。用户只需编写 map() 和 reduce() 两个函数，即可完成简单的分布式程序的设计。

map() 函数以 key/value 对作为输入，产生另外一系列 key/value 对作为中间输出写入本地磁盘。MapReduce 框架会自动将这些中间数据按照 key 值进行聚集，且 key 值相同（用户可设定聚集策略，默认情况下是对 key 值进行哈希取模）的数据被统一交给 reduce() 函数处理。

用户编写完 MapReduce 程序后，按照一定的规则指定程序的输入和输出目录，并提交到 Hadoop 集群中。作业在 Hadoop 中的执行流程如图 1 所示。Hadoop 将输入数据切分成若干个 split 片，并将每个 split 交给一个 Map Task 处理；Map Task 不断地从对应的 split 中解析出一个个 key/value，并调用 map() 函数处理，处理完之后根据 Reduce Task 个数将结果分成若干个分片（partition）写到本地磁盘；同时，每个 Reduce Task 从每个 Map Task 上读取属于自己的 partition，然后使用基于排序的方法将 key 相同的数据聚集在一起，调用 reduce() 函数处理，并将结果输出到文件中。

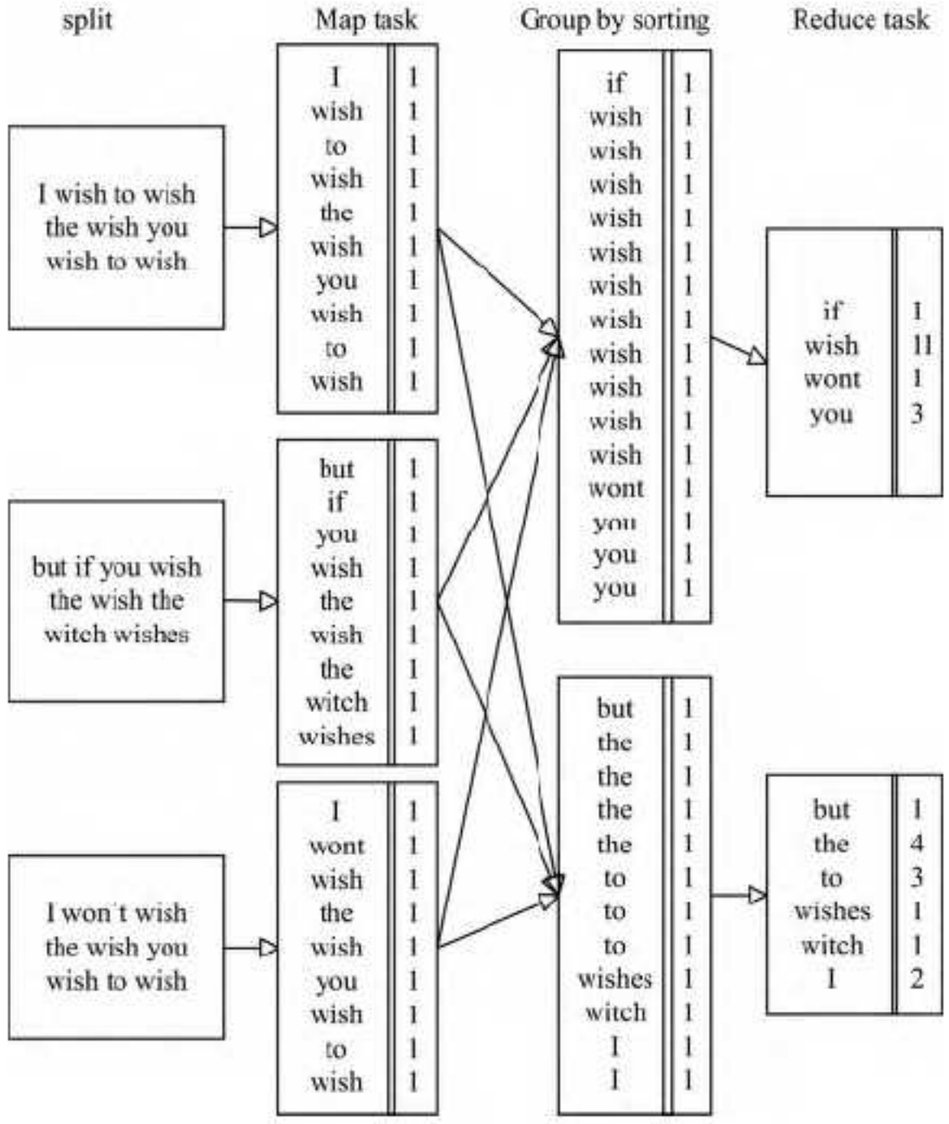

图 1 MapReduce 编程模型流程示例

3.2 编程模型改进方面的研究

最初，谷歌公司提出 MapReduce 的目的是配合公司自身的业务需要，即网页检索相关的问题。但是随着 MapReduce 的使用越来越广泛，其目的也不局限在搜索领域。这也使 MapReduce 的不足逐渐显露出来。为了提高 MapReduce 在不同的应用背景下的性能，很多学者对此进行了大量的研究，并加以改进，如表 1 所列。

表 1 几种 MapReduce 模型比较

模型名称	模型特点	缺点
Barrier-less	增加 Reduce 函数功能，使之能处理中间结果排序	增加编程负担
MapReduceMerge	在 Reduce 阶段后增加 Merge 过程，整合异构数据集结果	不支持迭代计算
Dache	增加 Dache 过程对框架进行缓存管理	在处理小规模作业量时效果不明显
MARCO	在 Map 过程中优先处理部分 Map 任务结果	在小规模集群中效果不明显

在 MapReduce 进行到 Sorting 阶段时，大量的中间结果要进行传输和排序，如图 1 中的 group by sorting，此时会消耗大量的时间。为了解决这个问题，文献[3]提出了 Barrier-less MapReduce 编程模型，该模型修改了 Reduce 函数，增强了它的功能，使之能够处理中间值 Key/Value 对来解决大量中间结果的排序问题。但是这个方法加重了用户变成的负担，用

户在编写 Map 和 Reduce 程序时需要对 Reduce 程序进行对应的修改才能达到上述要求。

Google 的 MapReduce 具有很强的处理同构数据集的能力，但是在处理异构数据集方面缺乏支持。为了使 MapReduce 具有处理异构数据集的能力，文献[4]提出了 MapReduceMerge 并行编程模型。该模型能够对多个异构数据集同时进行 Map 和 Reduce 操作。之后增加了一个 Merge 阶段，用来对之前的 Reduce 阶段产生的结果数据进行整合，得到全部的异构数据集的结果。

MapReduce 框架在运行时会产生大量的中间数据，在工作节点需要这些数据时，庞大的中间数据的传输问题就变得十分复杂。程序经常会被移动到数据节点上以进行数据的本地处理，然而这并不总是可行的，因为工作节点的身份并不是那么容易变换。数据本地性是另一个概念。文献[5]设计了一个面向大数据应用的数据感知缓存框架 Dache。它负责 MapReduce 运行时的缓存管理，任务会提交它们的中间结果给 Dache 负责的缓存管理器。其他任务在运行前会询问缓存管理器。缓存管理器本身也会收集工作节点可能会需要的数据，以使传输延迟和开销最小化。同时，Reduce 任务也可以利用 Map 阶段的缓存结果，使 MapReduce 作业的执行得到加速。

MapReduce 提供的自动数据管理和容错能力，提高了集群的可编程性。MapReduce 的执行模型中包括一个称为洗牌的从所有 Map 到所有 Reduce 阶段的通信，整个网络平分通信开销。有些 MapReduce 需要移动大量数据，强调对分带宽，并带来显著的运行时开销。因此这种优化重洗牌 MapReduce 是重要的，因为：

(1)它们包括关键应用（如倒排索引的搜索引擎和数据聚类机器学习）；

(2)它们运行时间超过原 MapReduce5 倍甚至更长。

MapReduce 的洗牌结果在洗牌和 Map 之间有一些重叠的异步特性。然而这种覆盖在洗牌负担很重的 MapReduce 中是不充分的。文献[6]提出的 MapReduce 与通信重叠(MARCO)通过与 Reduce 重叠的新颖想法，实现了几乎完全重叠。由于 MapReduce 只有在收到所有的 Map 数据后才开始 Reduce 计算的执行特性，MARCO 优先运行部分 Reduce 任务来处理一些 Map 任务的部分数据，而与其他 Map 任务的沟通重叠。MARCO 实现了隐藏重洗牌 MapReduce 中高洗牌量的延迟，以提升系统的基本性能。通过在 Hadoop 的 MapReduce 上实现 MARCO 的实验显示，一个 128 个节点的亚马逊 EC2 集群上，MARCO 的重洗牌 MapReduce 达到超过 Hadoop23%的平均增速。

4 对 MapReduce 调度器的优化

4.1 MapReduce 常见的调度策略

随着 MapReduce 的流行，其开源实现 Hadoop 也变得越来越受推崇。在 Hadoop 系统中，调度器非常重要，其作用是将系统中空闲的资源按一定策略分配给作业。在 Hadoop 中，调度器是一个可插拔的模块，用户可以根据自己的实际应用要求设计调度器。Hadoop 中常见的调度器有 3 种，分别为：

(1)默认的调度器 FIFO

Hadoop 中默认的调度器先按照作业的优先级高低，再按照到达时间的先后选择被执行的作业。它的缺点是不利于短小作业的处理。如果这个小作业排在一个大作业之后，小作业的作业响应时间大大增加。随着 Hadoop 的迅速普及，在单个 Hadoop 分布式集群中的用户数量、作业数量以及应用程序的种类都在不断增加。在复杂多变的作业批处理场景下，FIFO 的调度策略不能高效地利用集群计算资源，也逐渐满足不了不同作业背景的计算服务质量要求，其他更适用于集群复杂计算环境的多用户作业调度器被设计出来。

(2)计算能力调度器 Capacity Scheduler

它支持多个队列，每个队列可配置一定的资源量，每个队列采用 FIFO 调度策略，为了防止同一个用户的作业独占队列中的资源，该调度器会对同一用户提交的作业所占资源量进行限定。调度时，首先按以下策略选择一个合适队列：计算每个队列中正在运行的任务数与其应该分得的计算资源之间的比值，选择一个比值最小的队列，然后按以下策略选择该队列中一个作业：按照作业优先级和提交时间顺序选择，同时考虑用户资源量限制和内存限制。

(3)公平调度器 Fair Scheduler

Fair Scheduler 是由 Facebook 开发的多用户调度器。同计算能力调度器类似，它以资源池为单位划分计算资源，每个资源池可以设定资源的使用上限和最低保证。支持多队列多用户，每个队列中的资源量可以配置，同一队列中的作业公平共享队列中所有资源。

除了调度模型，在早期还有不少学者在对 Hadoop 调度器改进和优化中提出了很多经典的调度算法思想，近年来被借鉴和改进，如 Matei Zaharia 等人研究的 LATE 算法[20]的推测执行的思想、腾飞等人研究的 SPS 算法[21]的截止期调度思想等，以及考虑数据本地性和 Map 与 Reduce 关联性的 DELAY 调度算法[22]。

4.2 MapReduce 调度器优化方面的研究

MapReduce 集群的性能在很大程度上与调度器的算法有关，根据不同的特性的作业选择合适的调度器十分重要。目前很多学者也对 MapReduce 的调度器进行了研究。主要包括以下方面。

在使用传统的 Fair Scheduler 调度算法时，Map 任务通常比较小并且独立，容易并行化，而 Reduce 任务通常很长，并且直到任务结束时才释放已占用的资源。Map 和 Reduce 两个阶段的复杂性和独立性会产生重复观察到的饥饿问题。为缓解这一问题，文献[7]设计了耦合调度器，通过耦合进程，共同调度 Map 和 Reduce 任务而不是分隔地处理不同阶段的任务。为分析耦合调度算法的性能，作者设计了一个模仿 MapReduce 基础调度特性的模型，并通过仿真和实验，用泊松过程模拟了 10000 个作业池的处理场景，验证了对于 Map 阶段总时间均匀变化的一类作业，使用耦合调度算法的处理时间要优于使用传统的 Fair Scheduler 算法。但是耦合调度器的应用背景局限在了 Map 阶段总时间均匀变化的作业环境，在复杂的实际应用环境中调度器对集群作业效率的提高还需要更多的实验去证明。

通常对 MapReduce 并行编程模型的优化方案都是针对 MapReduce 过程中的某一个阶段，比如 Map、Reduce 或者 Shuffle 阶段。文献[8]提出了一种联合调度机制和一个明确

考虑 Map 和 Reduce 操作之间依赖关系的公式，描述了一个用来保证近似算法的常量参数。这个近似算法解决了指数约束的线性编程问题，并描述了一个基于逼近的队列生成器来解决线性编程松弛的问题。针对 map-shuffle-reduce 3 个阶段的联合调度提出了一个常量参数为 0 的近似算法(MASHERS)，并在 30 个处理器的环境下，使用每个作业包括 8 个 Map 任务和 3 个 Reduce 任务的 5 至 40 个作业的队列验证了算法的性能。

在大规模 Hadoop 集群中，数据的本地性对于性能的影响是十分重大的，这主要体现在大规模数据平台上使计算靠近数据的策略。在临近存储数据的地方调度任务可以有效地减少网络传输的开销，这对系统的稳定性和效能有很大的提高。虽然 Map 任务的数据本地性问题已经被发现并广泛研究，但是很多现有的调度算法忽视了在处理中间数据时 Reduce 任务的数据本地性问题，从而导致系统性能的下降。关于 Reduce 任务的数据本地性问题的重要性最近才被发现。然而现有的解决方案专门基于一个贪婪手段，依据直觉将 Reduce 任务放置在距离大部分中间数据较近的计算资源上。导致的结果是，随着作业序列的到达和离开，调度器把当前任务分配给当前开销最低的节点以保持本地性，却阻止了后续任务分配到这个开销更小却已经被占用的节点。文献[9]设计出了一个随机优化框架来提升 Reduce 任务中的数据本地性，并通过仿真和实验验证了性能的提高。

同样是针对数据本地性的优化，文献[10]提出了一个随机网络的观点。为同时实现最大的吞吐量和最小的延迟去调节数据本地性和负载均衡之间的平衡，文献提出一个新的队列结构和一个“加入最短队列”的 Map 任务调度策略算法，并且确定了在能力区域的一个外在约束，证明了算法在这个外在约束条件下对于任何到达速率的稳定性。事实证明，算法优化了吞吐量和外在约束在能力区域的一致性。作者还研究了在该算法下积压任务的数量，证明了该算法在重负载情况下的性能，算法在到达速率向量接近能力界限时逐渐地减少了积压任务的数量，使系统在重负载条件下达到了一个最理想的延迟。但是“最短队列”有可能不是“时间最短队列”。因此在合理地选择“最短队列”时也应考虑到作业类型(CPU 密集型和 I/O 密集型)，以便选出更加合理的队列。

现有的调度器已经能够满足一般的作业需要，但是仍然有提升的空间。虽然在 Map 任务和 Reduce 任务之间存在着很强的关联性，但它们没有被共同优化，这可能会导致作业饥饿问题或者不顺利的数据本地化。文献[11]设计出了一种具有资源感知能力的 MapReduce 调度器，它把对 Map 任务和对 Reduce 任务的处理连接起来，利用 Reduce 任务的等待调度和 Map 任务的随机调度共同优化任务配置。这种调度配置可以缓和饥饿问题，提升数据本地性。经过广泛的实验证明这种调度算法能够提高作业的响应速度。

目前最近的几篇关于 MapReduce 的论文大都侧重于 MapReduce 调度算法的优化，很少有关于 MapReduce 环境的理论研究。文献[12]使用了一种抽象方法来解决这个问题，简要地抽象并表述出 MapReduce 的调度模型。通过大量的在线和离线方式的实验改进模型并减少 MapReduce 任务的完成时间。由于完全的优化方案是 NP 难问题，作者提出了一种基于 3 因素的近似优化算法，通过仿真比较多种 MapReduce 的标准调度算法，如 FIFO、最短作业优先等，证明了这种近似算法优于已有的标准算法。

5 MapReduce 具体应用方面的研究

5.1 基于 Hive 和 Pig 的优化

MapReduce 的具体应用主要体现在基于 MapReduce 系统的处理实际问题的应用实现，其中应用最广泛的是 Hive 和 Pig 工具。

Pig 是一种编程语言，它简化了 Hadoop 常见的工作任务。Pig 可加载数据、表达转换数据以及存储最终结果。Pig 内置的操作使得半结构化数据变得有意义(如日志文件)。同时 Pig 可扩展使用 Java 中添加的自定义数据类型并支持数据转换。

Hive 在 Hadoop 中扮演数据仓库的角色。Hive 将数据的结构添加在 HDFS 中(hive superimposes structure on data in HDFS)，并允许使用类似于 SQL 语法进行数据查询。与 Pig 一样，Hive 的核心功能是可扩展的。

Hive 更适合于数据仓库的任务，主要用于静态的结构以及需要经常分析的工作。Hive 与 SQL 的相似促使其成为 Hadoop 与其他 BI 工具结合的理想交集。Pig 赋予开发人员在大数据集领域更多的灵活性，并允许开发简洁的脚本用于转换数据流以便嵌入到较大的应用程序。Pig 相比 Hive 相对轻量，它主要的优势是相比于直接使用 Hadoop Java APIs 可大幅削减代码量。

Hive 和 Pig 的广泛应用表明，在 MapReduce 成为大型集群中高效的大数据分析工具的同时，SQL 类查询语法在使用者与系统的交互中也起到了重要作用。通过 FaceBook 的日常运行结果显示，目前最常用的 SQL-to-MapReduce 转换工具 Hive 在执行特定类查询时速度是极其慢的。文献[13]认为，由于复杂的 SQL 结构和简单的 MapReduce 框架的不匹配，导致了现有的 SQL-to-MapReduce 转换器以一个操作对应一个作业的模式运行，并没有考虑到查询之间的相关性，因此限制了 MapReduce 对特定查询的执行性能。作者设计了一个名为 YSmart 的系统，该系统是一个具有相关性感知的 SQL-to-MapReduce 转换器。与现有的转换器相比，YSmart 可以应用一组规则来使用最少的 MapReduce 作业数去执行大量的复杂问询的相关操作，以显著减少冗余的计算、I/O 操作和网络传输。在 Amazon EC2 集群和 FaceBook 集群中对 YSmart 进行的复杂问询的集中评估，显示出 YSmart 的问询执行性能胜过 Hive 和 Pig 性能的 4 倍。

5.2 基于 HBase 的优化

HBase 是一个分布式的、面向列的开源数据库，该技术来源于 Bigtable[23]。就像 Bigtable 利用了 Google 文件系统(File System)所提供的分布式数据存储一样，HBase 在 Hadoop 之上提供了类似于 Bigtable 的能力。HBase 是 Apache 的 Hadoop 项目的子项目。不同于一般的关系数据库，HBase 是一个适合于非结构化数据存储的数据库。

随着数据的探索在近年来迅速增加，数据存储和数据处理越来越多地关注于提取重要信息。Hadoop 固有的可扩展性和容错性使之成为一个有吸引力的处理海量数据的并行解决方案，来处理在任一关系数据库系统或新兴的 NoSQL 数据库中的大型数据。以往大多数的研究重点集中于 SQL 或

类 SQL 的查询翻译与 Hadoop 的分布式文件系统，然而却难以在这样的文件系统中经常更新数据。因此需要一种不仅能横向扩展存储系统中的数据，而且还以透明的方式操纵可变数据的灵活的 HBase 数据存储系统。但是，HBase 的接口对大多数用户来说不是足够友好。SQL 客户端应用程序和数据库的连接组成的 HBase 的一个 GUI 将缓解学习曲线。文献[14]提出了 JackHare 框架的 SQL 查询编译器 JDBC 驱动程序和使用 MapReduce 框架在 HBase 中处理非结构化数据的系统化方法。导入 JDBC 驱动程序到 SQL 客户端 GUI 后，就可以利用 HBase 作为底层数据存储来执行 ANSI-SQL 查询。实验结果表明，此方法效率和可伸缩性表现良好。

5.3 对数据中心级别 MapReduce 集群的优化

出于对功耗、成本、性能和其他原因的考虑，数据中心级别的集群正朝着硬件异构的趋势发展。MapReduce 作为知名的编程模型，在处理集群中海量数据时经常被引用。有很多的 MapReduce 的实现在设计和优化上是针对异构集群的，但是其性能很不理想。比如在由 10 个志强服务器和 80 个灵动服务器组成的 90 节点的异构集群上，MapReduce 的性能表现不如其在 10 个志强服务器或 80 个灵动服务器组成的同构子集群上。

文献[15]提出，导致 MapReduce 在异构集群中性能下降的原因主要有两点。一是由于负载均衡，在 Map 阶段会有过重的和突发的网络通信；二是由于在 Reduce 阶段异构结构本身导致负载的不均衡被放大。文献[15]设计出一种名为 Tarazu 的优化系统用以优化 MapReduce 在异构集群中的性能，Tarazu 主要包括以下几个特点：

1. 具有流量感知的节点间 Map 阶段的负载均衡器；

2. Map 计算的流量感知调度器可以避免突发的网络传输；

3. Reduce 计算的预负载均衡处理。

作者在 90 节点的异构集群上的实验，验证了 Tarazu 的优化使得 Hadoop 的性能得到了提高。已经可以成功地使用基于 GPU 的计算机进行科学计算，依赖分布式节点集群来突破设备内存的限制。但是只有一小部分的文字挖掘的应用从这种结构中获益。由于在初始化阶段需要处理的是典型的数据密集型作业，易于调度的算法是提升应用性能的重要因素。文献[15]设计出了一种灵活的、基于 MapReduce 的分布式的文字挖掘应用的工作流，该工作流可以在 CPU 上执行 I/O 密集型操作也可以在 GPU 上执行经过优化的计算密集型操作，优化的主要作用是确保合并内存的正确性以及有效地使用共享内存。经过在由两块 NVIDIA Tesla M2050 组成的 8 节点集群中的大量实验，证实系统达到了预想的加速效果。

结束语　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进行了卓有成效的研究。随着网络中数据的爆发式增长，对海量数据处理模型的渴求越来越强烈。预计在今后的一段时间内，与 MapReduse 编程模型相关的研究会朝着以下方向进行：

(1)单计算节点性能

谷歌设计 MapReduce 的初衷是和自己本身的业务密切相关的，即网页爬图以及海量网页的存储和索引，同时也希望节约成本，利用自己的中低性能的廉价服务器作集群。但是在摩尔定律的作用背景下计算机已经发展了十几年，集群单节点的计算能力也在提高。如何让最初为廉价节点设计的集群框架在高性能节点上发挥应有的性能是一个值得研究的问题。

(2)基于任务类型感知的调度器优化

在 MapReduce 集群中每一时刻都有大量作业在同时运行，集群计算的时间开销中 70% 为传输开销。所以合理的调度器设计对于 MapReduce 集群的性能提升至关重要。通常每一种调度策略对应了一种任务背景，当处理另一种任务时，之前的调度器的性能很可能得不到充分发挥。所以通过感知任务类型来调整调度策略可以提高集群计算效率。

在云计算越来越贴近人们生活的今天，MapReduce 编程模型的出现简化了开发人员进行并行程序开发和批数据处理的流程，也使海量数据处理和存储在大数据背景的今天更贴近普通的程序员。MapReduce 也必将在现代互联网发展中发挥越来越重要的作用。

参 考 文 献

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

[2] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. ACM SIGOPS Operating Systems Review, ACM, 2003, 37(5): 29-43

[3] Verma A, Cho B, Zea N, et al. Breaking the MapReduce stage barrier[J]. Cluster computing, 2013, 16(1): 191-206

[4] Yang H, Dasdan A, Hsiao R L, et al. Map-reduce-merge: simplified relational data processing on large clusters[C]// Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. ACM, 2007: 1029-1040

[5] Zhao Y, Wu J. Dache: A data aware caching for big-data applications using the MapReduce framework[C]// 2013 Proceedings IEEE INFOCOM. IEEE, 2013: 35-39

[6] Ahmad F, Lee S, Thottethodi M, et al. MapReduce with communication overlap (MaRCO)[J]. J. Parallel Distrib. Comput. JPDC, 2013, 73(5): 608-620

[7] Tan J, Meng X, Zhang L. Performance analysis of coupling scheduler for mapreduce/hadoop[C]// 2012 Proceedings IEEE INFOCOM. IEEE, 2012: 2586-2590

[8] Chen F, Kodialam M, Lakshman T V. Joint scheduling of processing and shuffle phases in mapreduce systems[C]// 2012 Proceedings IEEE INFOCOM. IEEE, 2012: 1143-1151

[9] Tan J, Meng S, Meng X, et al. Improving ReduceTask data locality for sequential MapReduce jobs[C]// 2013 Proceedings IEEE INFOCOM. IEEE, 2013: 1627-1635

[10] Wang W, Zhu K, Ying L, et al. Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality [C]// 2013 Proceedings IEEE INFOCOM. IEEE, 2013: 1609-1617

[11] Tan J, Meng X, Zhang L. Coupling task progress for mapreduce resource-aware scheduling[C]// 2013 Proceedings IEEE INFOCOM. IEEE, 2013: 1618-1626

[12] Chang H, Kodialam M, Kompella R R, et al. Scheduling in mapreduce-like systems for fast completion time[C]// 2011 Proceedings IEEE INFOCOM. IEEE, 2011: 3074-3082

(下转第 564 页)

发展过程中，二者的发展规模和速度可能会出现两种情形。一种可能是领域分析将会呈现出二者平行发展的趋势，另一种可能是行业化成为领域分析研究的主干，一般化成为辅助部分。

不管怎样，仍然需要有一个统一的理论体系囊括这两个分支，探索和发现一些一般性的原理和方法，形成领域分析的基本原理。目前，领域分析还没有形成一套完整的理论，很需要也很有必要逐渐形成领域分析的独立的理论体系和研究方法，把领域分析作为一门学科来建设和开发，建立领域分析学。

在领域分析的发展过程中，领域分析的概念不断演变。今天我们认为，领域分析是对软件开发所涉及到的领域内的概念、方法进行总结和研究的活动的总称，这些活动的目标是建立领域模型，这些活动包括对领域模型中将要使用的概念、元素、实体、方法、映射等模型构件的抽象和约简。领域分析是软件分析中的一项创新活动，它可以找到和发现新的概念或者方法，或者映射等，从而可以改进和优化软件的设计和应用项目的实现，甚至可以对软件开发方法促成突破性的变革，因此，领域分析是软件开发过程中的一个十分重要的环节。软件分析师应该更多地致力于领域分析。

事实上，领域分析需要创造性的思维，是一种创新活动。未来的领域分析应该以创新为主旋律。在可预见的很长一段时间内，领域分析仍然是软件分析的可靠基础。目前，诸如云计算、普适计算和社会计算等新技术不断涌现和发展起来，给领域分析提出了新的问题和挑战。领域分析要解决这些新的问题，就应该在应对这些新技术变革带来的不相适应、不相配套等问题的过程中，全面调整自己的理论框架和方法，形成一套在软件开发中可以有效使用的领域分析技术和方法。

参 考 文 献

[1] Neighbors J M. Software construction using components[D]. University of California, Irvine, 1980

[2] Arango G, Prieto-Diaz R. Domain analysis concepts and research directions[M]. Domain Analysis and Software Systems Modelling. IEEE Computer Society, Washington DC, 1991: 9-33

[3] De Champeaux D, Lea D, Object-oriented system development [M]// Reading. MA: Addison-Wesley, 1993

[4] Czarnecki K, Eisenecker U W. Generative programming [M]. Addison-wesley Professional, 2000

[5] Czarnecki K. Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models[OL]. http://citeseerx.ist.psu.edu/showciting?cid=106655

[6] Evans E. Domain-driven design: tackling complexity in the heart of software[M]. Boston: Addison-Wesley Professional, 2004

[7] Bushmann F, Meunier R, Rohnert H, et al. Pattern-oriented software architecture: A system of patterns [M]. John Wiley & Sons, 1996

[8] Buschmann F, Henney K, et al Pattern-oriented Software Architecture: a Pattern Language for Distributed Computing, Volume 4[M]. New York: John Wiley & Sons, 2007

[9] Buschmann F, Henney K, Schmidt D C. Pattern-oriented software architecture: On patterns and pattern languages, vol. 5 [M]. 2007

[10] Bjørner D. Domain Analysis: Endurants-An Analysis & Description Process Model[C]// Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi. Springer, 2014

[11] Bjørner D. Domain Analysis & Description: Perdurants [R]. DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Fall/Winter, 2014

[12] Bjørner D. Domain Endurants[M]//Specification, Algebra, and Software. Springer Berlin Heidelberg, 2014: 1-34

[13] Wu Bao-ming. General analysis of requirements for risk-oriented financial data modeling[C]// Proceedings-2010 2nd International Conference on Modeling, Simulation, and Visualization Methods, WMSVM 2010. Sanya, Hainan, China, 2010: 46-49

(上接第 541 页)

[13] Lee R, Luo T, Huai Y, et al. Ysmart: Yet another sql-to-mapreduce translator[C]// 2011 31st International Conference on Distributed Computing Systems (ICDCS). IEEE, 2011: 25-36

[14] Chung W C, Lin H P, Chen S C, et al. JackHare: a framework for SQL to NoSQL translation using MapReduce[J]. Automated Software Engineering, 2014, 21(4): 489-508

[15] Ahmad F, Chakradhar S T, Raghunathan A, et al. Tarazu: optimizing MapReduce on heterogeneous clusters [J]. ACM SIGARCH Computer Architecture News, ACM, 2012, 40(1): 61-74

[16] Zhang K, Chen X. Large-scale Deep Belief Nets with MapReduce [J]. Aceess, IEEE, 2014, 2: 395-403

[17] Li F, Ooi B C, Özsu M T, et al. Distributed data management using MapReduce[J]. ACM Computing Surveys (CSUR), 2014, 46(3): 31

[18] Zou Q, Li X B, Jiang W R, et al. Survey of MapReduce frame operation in bioinformatics[J]. Briefings in bioinformatics, 2014, 15(4): 637-647

[19] Qian J, Miao D, Zhang Z, et al. Parallel attribute reduction algorithms using MapReduce[J]. Information Sciences, 2014, 279: 671-690

[20] Zaharia M, Konwinski A, Joseph A D, et al. Improving MapReduce Performance in Heterogeneous Environments[J]. OSDI, 2008, 8(4): 29-42

[21] Teng F, Yang H, Li T, et al. Scheduling real-time workflow on mapreduce-based cloud[C]// Innovative Computing Technology (INTECH), 2013 Third International Conference on. IEEE, 2013: 117-122

[22] Zaharia M, Borthakur D, Sarma J S, et al. Job scheduling for multi-user mapreduce clusters[R]. UCB/EECS-2009-55. EECS Department, University of California, Berkeley, 2009

[23] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[C]// Proceeding of Conference on Usenix Symposium on Operating System Design and Implementation. 2006: 205-218

[24] 董西成. Hadoop 技术内幕 [M]. 北京: 机械工业出版社, 2013

[25] 李建江, 崔健, 王聃, 等. MapReduce 并行编程模型研究综述 [J]. 电子学报, 2012, 39(11): 2635-2642

[26] 吴煜祺, 曾国荪, 曾媛. 云计算环境下调度算法的趋势分析 [J]. 微电子学与计算机, 2012, 29(9): 103-108