Spark性能优化技术研究综述

廖湖声[1] 黄珊珊[1] 徐俊刚[2] 刘仁峰[2]
(北京工业大学信息学部 北京 100124)[1] (中国科学院大学计算机与控制学院 北京 101408)[2]

摘 要 近年来,随着大数据时代的到来,大数据处理平台发展迅速,产生了诸如Hadoop,Spark,Storm等优秀的大数据处理平台,其中Spark最为突出。随着Spark在国内外的广泛应用,其许多性能问题尚待解决。由于Spark底层的执行机制极为复杂,用户很难找到其性能瓶颈,更不要说进一步的优化。针对以上问题,从开发原则优化、内存优化、配置参数优化、调度优化、Shuffle过程优化5个方面对目前国内外的Spark优化技术进行总结和分析。最后,总结了目前Spark优化技术新的核心问题,并提出了未来的主要研究方向。

关键词 Spark,开发原则优化,参数优化,内存优化,调度优化,Shuffle过程优化
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.07.002

Survey on Performance Optimization Technologies for Spark

LIAO Hu-sheng[1] HUANG Shan-shan[1] XU Jun-gang[2] LIU Ren-feng[2]
(Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China)[1]
(School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing 101408, China)[2]

Abstract In recent years, with the advent of the era of big data, big data processing platform is developing very fast. A large number of big data processing platforms, including Hadoop, Spark, Strom and etc., have appeared, among which Apache Spark is the most prominent one. With the wide applications of Spark at home and abroad, there are many performance problems to be solved. As the underlying implementation mechanism of Spark is very complex, it is difficult for ordinary users to find performance bottlenecks, let alone further optimization. In light of the above problems, the performance optimization technologies for Spark were summarized and analyzed from five aspects, including development principles optimization, memory optimization, configuration parameter optimization, scheduling optimization and shuffle process optimization. Finally, the key problems of Spark optimization technologies were summarized and future research issues were proposed.

Keywords Spark, Development principle optimization, Configuration parameter optimization, Memory optimization, Scheduling optimization, Shuffle process optimization

1 引言

近年来,随着科技的进步与发展,从企业运作到互联网平台,再到各类科技设备,大量的数据源时刻产生着有价值的数据流。面对日益增长的数据量和快速处理的要求,单机的处理能力和I/O性能显然已经不再适用。因此,越来越多的应用向分布式系统发展;同时,越来越多的研究组织尝试将计算能力扩展到成百上千台机器上。随着研究的深入,研究者们发现,在很多领域中,数据处理所需的速度和复杂度也在逐渐增加,例如,除了简单的查询,机器学习和图分析这类需要多次迭代的复杂算法也得到了广泛的应用;同时,一些实时数据源上的流分析以保证能够及时获取有效信息也是企业需要的功能。因此,针对上述问题,研究者们提出了一种大型集群上的快速和通用数据处理架构Spark,它能解决多数数据处理作业的需求,且具有很强的扩展性[1]。

Spark是由美国加州大学伯克利分校AMPLab提出的一个大数据分析平台,其特点是基于内存进行计算且提出了弹性式分布数据集[2](Resilient Distributed Dataset,RDD)的概念,不但可以多迭代批量处理数据,还可以兼顾数据仓库、流处理和图计算等多种范式,是大数据系统领域的全栈计算平台[3]。由于Spark具有出色的数据处理能力和高扩展性,众多企业已经在实际生产中进行了推广和应用。例如,Yahoo将Spark用于Audience Expansion中,以更准确地通过广告寻找目标用户;百度推出了大数据处理产品BMR(Baidu MapReduce);腾讯推出了广点通等。这些企业都拥有庞大的Spark集群,例如,腾讯的Spark集群已经达到千台的规模。

到稿日期:2017-07-01 返修日期:2017-08-15 本文受国家自然科学基金项目:云中并行程序性能分析方法研究(61372171)资助。
廖湖声(1954-),男,博士,教授,主要研究领域为软件自动化方法、数据集成技术;**黄珊珊**(1992-),女,硕士,CCF学生会员,主要研究领域为分布式计算框架的性能优化方法,E-mail:huangss118@emails.bjut.edu.cn(通信作者);**徐俊刚**(1972-),男,博士,教授,CCF会员,主要研究领域为大数据管理和深度学习;**刘仁峰**(1993-),男,硕士,CCF学生会员,主要研究领域为大数据系统管理和大数据平台性能分析与优化方法。

随着 Spark 在国内外的广泛应用，其在实际应用中的一些问题也逐渐暴露。一个最主要的问题就是 Spark 的性能问题，大数据平台的执行环境由于受到底层硬件、体系结构、操作系统、Spark 框架本身以及用户编写的应用程序等多层次的综合影响，用户实际应用过程中很难使其达到理论的性能峰值。当面对 Spark 平台的性能问题时，用户因不了解 Spark 这种分布式计算平台底层复杂的执行机制，对找到性能瓶颈并进一步优化几乎束手无策。

笔者通过大量调研发现，目前国内针对 Spark 平台性能优化的研究刚刚起步，仅有国外少量学者对其进行了研究。基于此现状，本文较为全面地整理和分析了国内外关于 Spark 优化技术的研究资料，为相关领域学者的深入研究提供参考。

2 相关背景

在大数据计算领域，Spark 已经成为越来越流行的计算平台之一。Spark[4] 的功能涵盖了大数据领域的离线批处理、SQL 类处理、流式/实时计算、机器学习、图计算等各种不同类型的计算操作。Spark 也在近几年成为了 Apache 软件基金会最活跃的项目之一，众多开源社区也将 Spark 作为热门研究项目。

相比于其他大数据平台，Spark 具有以下优势：1）速度，与 Hadoop MapReduce[5] 相比，Spark 基于内存的运算速度要快 100 倍以上，如图 1 所示，而基于硬盘的运算速度也要快 10 倍以上；2）易用，Spark 支持 Java，Python，Scala 和 R 的 API，还支持超过 80 种的高级算法；3）通用性，图 2 为 Spark 包含的 Spark SQL，Spark Streaming，Spark MLlib 以及 Spark GraphX 组件，为企业基于统一的平台处理不同类型的数据提供了一站式的解决方案；4）融合性，Spark 可以与其他开源产品融合，如可以使用 Hadoop 的 YARN 和 Apache Mesos[6] 作为其资源管理和调度器，并且可以访问不同的数据，如 HBase[7]，HDFS，S3 和 Cassandra[8] 等。Spark 也可以运行在 Standalone 下，且不依赖第三方。

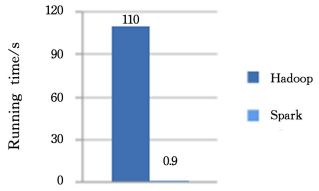

图 1 Spark 和 Hadoop 的逻辑回归运算速度的对比

Fig. 1 Comparison of computation speed of logistic regression between Spark and Hadoop

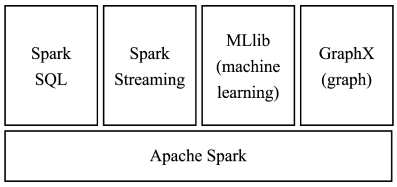

图 2 Spark 生态圈

Fig. 2 Spark ecosystem

然而，高效地利用 Spark 平台进行大数据处理并非易事。针对不同的业务场景和数据，需要对 Spark 作业进行不同的合理优化，否则 Spark 作业的执行速度可能会很慢，无法体现 Spark 计算引擎的优势。因此，如何有效地优化 Spark 的性能已成为当前 Spark 大数据处理领域的重要研究课题。

本文对近年来的分布式大数据处理技术进行了大量调研，图 3 给出了近 5 年来 Google 上 Apache Spark 与 Apache Hadoop 搜索量的趋势，可以看出人们对 Spark 的关注程度正在提升，而 Hadoop 则保持着一个较为平稳的态势。图 4 给出了国内外已发表的相关文献数量的趋势，下方虚线为国内核心期刊数据检索的文献数量趋势，上方实线为 Google 学术检索的文献数量趋势，显然，我国针对 Spark 以及 Hadoop 这种大数据处理平台的研究成果较少。因此，本文主要探讨 Spark 的性能优化技术。从一线开发技术人员提出的多种 Spark 作业开发优化原则入手，对现有的 Spark 性能优化技术进行总结。分别从以下 5 个方面提出目前 Spark 性能优化的研究趋势和热点：1）从 Spark 开发原则的角度提出 Spark 性能优化方案；2）基于 Spark 内存的运行原理，提出现有的性能优化方案；3）针对 Spark 作业运行参数的配置，提出针对不同业务场景的性能优化方案；4）针对 Spark 作业调度算法，综述并对比国内外的调度优化方案；5）针对 Spark 中的 Shuffle 模块，提出了针对 Shuffle 实现过程中不同方面的优化方案。

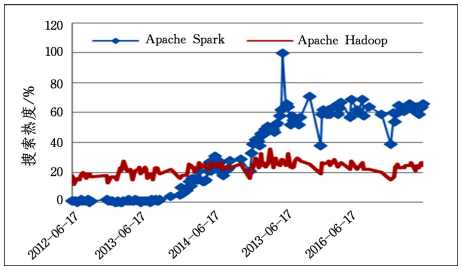

图 3 Google 每日搜索量

Fig. 3 Daily search volume of Google

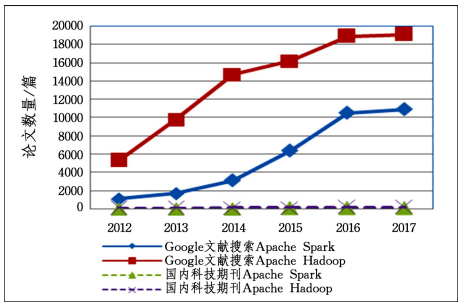

图 4 国内外相关文献

Fig. 4 Domestic and foreign related literatures

3 Spark 的工作原理

在了解如何优化 Spark 性能之前，需要简单介绍 Spark 的工作原理，以便理解针对不同 Spark 执行过程的优化方式。本节从 Spark 的系统运行架构和 Spark 的运行逻辑两方面来对 Spark 的工作原理进行说明。

3.1 Spark 系统的运行架构

Spark 的整体架构如图 5 所示。其中，Driver 是用户编写

的数据处理逻辑,Driver 运行应用程序的 main()函数并创建 SparkContext,SparkContext 是用户逻辑与 Spark 集群主要的交互接口,它会与 Cluster Manager 交互。

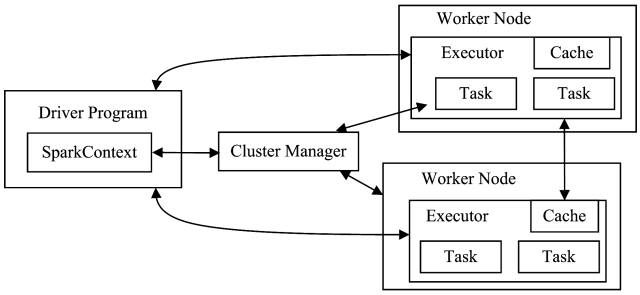

图 5　Spark 的整体架构

Fig. 5　Overall architecture of Spark

Cluster Manager 负责集群的资源管理和调度,现支持 Local,Standalone Deploy Mode,Apache Mesos,Hadoop Yarn 和 Amazon EC2 等。Cluster Manager 会根据用户提交任务时设置的 CPU 和内存等信息为本次提交分配计算资源,并启动 Executor。Worker Node 是集群中执行计算任务的节点,Task 是每个 Executor 上的计算单元。每个 Worker Node 会为应用启动一个 Executor 进程。Executor 会启动线程池,线程池负责管理 Task 的运行,同时 Executor 会将数据存储在内存或者磁盘上。执行 Task 后,Executor 会将 Task 的运行状态汇报给 Driver,在所有的 Task 都正确执行或者超过执行次数仍然没有执行成功时停止。

3.2　Spark 的运行逻辑

为了更好地说明 Spark 的运行逻辑,以 Spark 自带的实例 WordCount 为例来说明整个 Spark 的运行过程。WordCount 的代码如图 6 所示,这段代码表达了统计文档中不同单词的出现次数。程序的逻辑执行流程如图 7 所示,但 Spark 中程序的实际执行流程要复杂得多。

```
val textFile=sc.textFile("hdfs://…")
val counts=textFile.flatMap(line =>line.split(" "))
            .map(word => (word,1))
            .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://…")
```

图 6　Spark WordCount 的示例代码

Fig. 6　Sample code of Spark WordCount

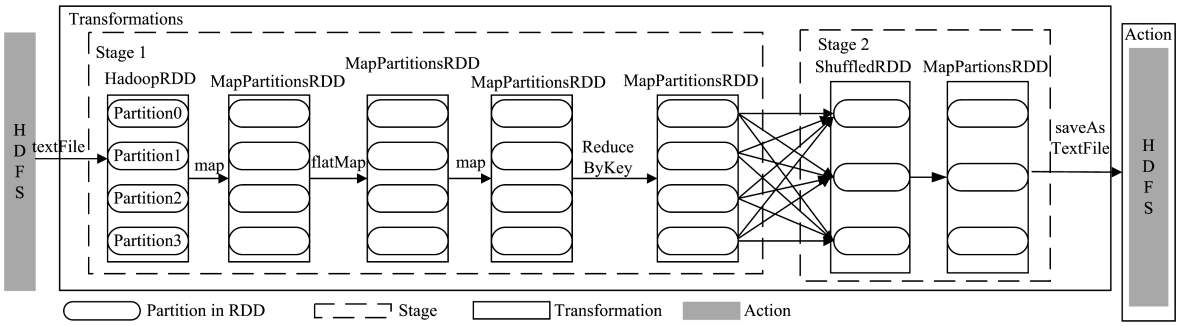

图 7　WordCount 程序的运行逻辑图

Fig. 7　Program running logic of WordCount

用户的 Driver 程序可以按 Action 操作划分 Job,并按照 Action 出现的先后顺序提交对应的 Job,即程序有多少个 Action 就有多少个 Job。WordCount 程序生成了一个 Job,在该程序中只有一个 Action 操作,即 saveAsTextFile()操作。

Job 提交后程序的运行逻辑如图 7 所示,在 Spark 应用中,整个执行流程在逻辑运算之前会形成有向无环图(Directed Acyclic Graph,DAG)。Action 算子触发后会将所有累计的算子形成一个有向无环图,然后由调度器调度该图上的任务进行运算。Spark 的调度方式与 MapReduce 有所不同。Spark 根据弹性分布式数据集 RDD 之间不同的依赖关系(窄依赖:子 RDD 只依赖于父 RDD(s)固定数量的 Partition;宽依赖:子 RDD 的每个 Partition 都依赖于父 RDD(s)所有的 Partition)切分形成不同的阶段(Stage)。在一个 Stage 内部,每个 Partition 都会被分配一个计算任务(Task),这些 Task 是可以并行执行的。图 7 中的长方形条框分别代表不同的 RDD,RDD 内部的椭圆形方框代表一个数据块。数据从 HDFS 读入 Spark,形成 HadoopRDD,经过一系列转换操作后遇到宽依赖 RDD,以此划分,进行 Shuffle 操作,即进入第二个 Stage。最后,MapPartitionRDD 通过函数 saveAsTextFile 输出并保存到 HDFS 中。

4　Spark 性能优化技术的研究进展

作为当前流行的并行计算架构,Google 提出的 MapReduce[9] 编程模型结构可以使用户专注于业务逻辑,而不必关心一些传统模型中需处理的复杂问题,例如并行化、容错、负载均衡等。与以前的并行计算系统相比,MapReduce 具有易用(只需要少量并行编程的概念就可以处理广泛的搜索问题)、扩展性好和硬件性价比高等优点。但是,MapReduce 并未做到极致,在实际开发过程中直接使用 MapReduce 编程仍然是一件十分困难的事情。此时,出现了基于 MapReduce 语言的上层封装 Pig[10] 和 Hive[11]。Pig 和 Hive 都是 Hadoop 上的开源实现,它们最大的不同在于语法上存在差异,其中 Pig 使用类似脚本语言的语法,而 Hive 使用的是类似 SQL 语言的语法。当 Pig 或 Hive 编写数据处理程序时,编译器会将其翻译成 MapReduce 代码,使其可以直接在 Hadoop 上运行。然而,随着需要处理的数据量的增长,人们对 MapReduce 的运算速度提出了更高的要求,对 MapReduce 性能的优化也成为了研究的热点。目前,对 Hadoop 的优化取得了不错的成果,如针对 Hadoop 在迭代计算方面的不足,华盛顿大学提出了 HaLoop[12-13],以高效地支持迭代,通过缓存循环不变量来

分析任务。Berkeley 大学提出了一种在集群中缓存数据的方案 PACMan[14]，PACMan 实现了两个缓存替换策略，针对数据密集型的并行任务进行优化，能最大限度地减少平均工作时间，提高集群效率。在并行数据库系统方面，Pavlo 等[15]通过将 MapReduce 与两个经典数据库系统 DBMS 和 Vertica 进行性能基准对比后发现，MapReduce 虽然在数据加载上的效率很高，但是在性能方面还是低于两个经典数据库系统。对此，新加坡国立大学研究了影响 MapReduce 框架的 5 个主要因素[16]，分别从 I/O 模型、数据解析、索引和调度 4 个方面对其进行优化，最后通过实验表明效率提升了 2.5 倍到 3.5 倍不等，最优配置可大幅缩减 MapReduce 和经典数据库系统之间的性能差距。由此可见，针对 MapReduce 的优化在一定程度上提升了运行效率，但由于模型本身的限制，运行速度的提升有限，只有突破模型的限制，才能大幅提高任务的运行速度。Spark 就是在这种环境下出现的一种新的并行计算系统。

不同于 MapReduce 基于磁盘进行大数据的批量处理，Spark 提出基于 RDD 的数据处理，其可以显式地将 RDD 数据存储到磁盘和内存中，不但有效改善了 MapReduce 在磁盘中存储数据时会产生大量的 I/O 消耗和网络传输消耗的问题，而且适合数据挖掘、机器学习等迭代式计算任务；同时，Spark 提供了如 map，filter，flatMap，reduceByKey，join，sort 等数据集操作类型，相较于只有 Map 和 Reduce 两种操作的 MapReduce，其提供了一种更灵活的编程模型。然而，Spark 计算框架对集群资源配置的要求较高，在资源调度方面也存在无法细粒度地进行资源分配的问题，因此如何提高 Spark 作业运行的性能并充分发挥其优势成为了研究的关键点。下面从开发原则、内存、配置参数、调度、Shuffle 过程 5 个方面总结和分析目前的优化研究方案。

4.1 开发原则优化

考虑 Spark 性能优化的第一步，就是在开发 Spark 作业的过程中注意和应用一些开发原则的优化。这些开发原则大多来源于 Spark 官网的 Tuning Spark 或者互联网公司的一线技术团队[17]。本文抽取了当前应用的主要优化原则并进行总结，如表 1 所列。此外，卞昊穹等[18]针对 Spark 上代价较高的等值连接操作进行了优化，将 Simi-Join 和 Partition Join[19-20]等常用连接算法的优势相结合，提出了一种改进的等值连接算法。实验表明，相比现有的基于 Spark 的数据分析系统的连接算法，该算法将性能提升了 1～2 倍。

表 1 总结了目前被大家所熟知的 Spark 开发优化原则，但是在实践中要满足所有原则具有相当大的难度，只有对 Spark 作业进行综合性分析并针对不同的场景采用不同的方案才能获得最佳的性能。对 Spark 作业的优化应该从多方面入手，不仅需要在开发过程中注意一些原则的应用，更需要从 Spark 本身的角度出发来进行优化，下文将继续展开描述。

表 1 开发原则的优化

Table 1 Optimization of development principle

规则	优化原理
避免创建重复的 RDD	对于一份数据，只应创建一个 RDD，对一份数据执行不同的算子操作，不能创建多个 RDD 来代表同一份数据
提高 RDD 的复用率	当多个 RDD 的数据有重叠或者包含时，可以通过提取 RDD 中需要的数据进行复用，而不是再创建 RDD
持久化多次重复使用的 RDD	通过 persist 或 cache 方法可以标记一个被持久化的 RDD，一旦被触发，该 RDD 会被保留在计算节点的内存中，之后每次对该 RDD 进行算子操作时，都会从内存中直接提取数据，而不会从源头重新计算
使用高性能算子	例如，使用 reduceByKey/aggregateByKey 替代 groupByKey；使用 mapPartitions 替代 map；使用 foreachPartition 替代 foreach；使用 filter 之后进行 coalesce 操作等
优化数据结构	避免使用一些增加额外开销的 Java 特性，例如使用对象数组以及原始类型(primitive type)数组代替 Java 或者 Scala 集合类(collection class)；尽可能避免采用含有指针的嵌套数据结构来保存小对象等方法

4.2 内存优化

Spark 作为基于集群的计算引擎，与传统的基于硬盘和网络的计算方式相比，使用内存对中间结果进行存储可以减少数据传输的时间，进而保证任务的运行效率。本节讨论了 Spark 对内存的使用行为，介绍了目前基于 Spark 内存的性能优化研究成果。Spark 的核心思想就是充分利用内存作为缓存来实现不同阶段之间的数据共享，加速程序的执行。不可避免的是，Spark 对内存的使用还处于初级和低效阶段，对内存的使用效率完全依赖于程序员所编写的代码的质量，因此针对 Spark 内存的优化是很有意义的。

文献[21-22]通过对内存行为的建模与分析以及对代码的语义分析，实现了内存策略的自动化，即调度器可以自动识别出有价值的数据集(RDD)并将其放入缓存，在避免缓存污染的同时，减轻了程序员的编程负担。同时，在对代码进行语义分析和获得任务详细信息的基础上，根据 RDD 大小和权重计算，进行操作顺序的优化重排；使用加了权重信息的寄存器分配模型作为新的替换算法来代替原有的 LRU 算法及多级缓存模型。通过以上两点优化，任务在资源有限情况下的运行效率以及在不同集群环境下任务效率的稳定性得到了提升。但该文献中提出的理论模型还不够完善，在 RDD 的权重计算准确性部分以及节点之间的内存共享和任务之间的内存共享部分均需要完善。Spark 基于内存的快速数据处理能力，使得一些算法的并行度和大数据适应能力得以提高。萨初日拉等[23]利用 Spark 的快速数据处理能力，可以减轻海量数据下数据立方体计算任务的压力，基于内存重复使用和共享的思想改进了数据立方体计算方法。首先，在 Spark 平台下实现了 BUC 算法，称为 BUCPark(BUC on Spark)，并针对 BUCPark 算法迭代次数过多的缺陷进行了改造，提出了 LBUCPark 算法(Layered BUC on Spark)。分析实验结果可以看出，该算法确实能大幅提高联机分析处理(Online Analytical Processing，OLAP)大数据时的实时响应性能。

4.3 配置参数优化

Spark 中共有近 180 个配置参数，这些配置参数共同控制着 Spark 作业的运行，用户可以根据自己的需求合理地调整这些配置参数。实验表明，Spark 系统配置参数的设置对

作业的运行性能(如内存分配和使用率、系统并行率、I/O 开销及网络带宽的使用率等)有很大的影响[24]。

通过第 3 节介绍的 Spark 的工作原理可知,当使用 spark-submit 提交一个 Spark 作业后,这个作业就会启动一个对应的 Driver 进程。根据所使用的部署方式(deploy-mode)的不同,Driver 进程可能在本地启动,也可能在集群中的某个工作节点上启动,Driver 进程本身会根据设置的配置参数占有一定数量的内存和 CPU core。Driver 会向集群管理器申请运行 Spark 作业需要使用的资源,这里的资源可被理解为 Executor 进程,集群管理器会根据 Spark 作业设置的资源参数在各个工作节点上启动相应的 Executor 进程,每个 Executor 进程都占有一定数量的内存和 CPU core。Executor 的内存主要分为 3 部分:一部分供 Task 执行用户自己编写的代码,默认占总内存的 20%;一部分用于 task 通过 Shuffle 过程去获取一个 Stage 的 Task 的输出,然后进行聚合等操作,也默认占总内存的 20%;最后一部分用于 RDD 的持久化,默认占总内存的 60%。因此,所谓的配置参数的优化,主要是在 Spark 运行过程中各个使用资源的位置调节默认配置参数,进而优化资源的使用率,提高 Spark 作业的性能。表 2 列出了几个对 Spark 作业性能的优化具有重大影响的配置参数,同时也总结了互联网公司的一线技术团队[17]给出的对配置参数的优化建议。

表 2　配置参数的优化

Table 2　Optimization of configuration parameter

参数	参数意义	调优建议
num-executors	Executor 进程数量	Executor 进程数量的多少直接影响程序的并行度,建议 50～100 个
executor-memory	Executor 进程内存	4～8 GB
driver-memory	Driver 进程内存	通常不设置,或者 1 GB 左右,若使用 collect 算子则应考虑增大 Driver 内存,以防止内存溢出
executor-cores	每个 Executor 进程占用的 CPU core 的数量	一般设置 2～4 个,因为每个 CPU core 只能执行一个 task 线程,因此每个 Executor 的 CPU core 数量越多,就能够越快速地执行完分配给自己的所有 Task 线程
driver-cores	Driver 进程占用的 CPU core 数量	核心数的多少对程序的并行执行有相当大的影响
spark.storage.memoryFraction	设置 RDD 持久化数据在 Executor 内存中的占用比	默认为 0.6,若有较多的 RDD 持久化操作,则适当提高该值

目前,已经有较多的学者对 MapReduce 框架上配置参数的优化进行了研究。Herodotou 等[25-27]提出了一个基于 Hadoop 平台的自动调优平台——Starfish,该平台能够针对 Hadoop 应用程序的不同执行流程进行优化。Wu 等[28-29]提出了一个基于应用程序分析和性能分析的 Hadoop MapReduce 集群配置优化框架(Profiling and Performance Analysis-based System,PPABS),该框架分为两个部分:1)分析器,根据最大可能提升 Hadoop 性能的参数,生成 MapReduce 应用程序的等价类;2)识别器,将进入 PPABS 的未知参数与其中一个等价类匹配,从而实现 Hadoop 配置参数的自我优化。文献[30]提出了一种基于机器学习的预测和优化搜索算法,实验表明该算法较传统方法可将性能提升 2～8 倍。但是,该模型还需要进一步细化,以保证适用于多种不同的负载。

与 MapReduce 不同,针对 Spark 系统进行配置参数优化的研究成果还很少。在参数优化方面,Ravi 等[31]在 Apache Big Data 2015 上提出了一种 Spark 参数配置和优化的方法;此外,Apache Spark 官方网站的 Tuning Spark 文档和 Cloudera 上关于 Spark 调优的有关博客也有参考意义。国内研究领域中,陈侨安等[32]提出了基于近邻搜索算法的 Spark 参数自动优化,主要通过分析任务的历史数据库和基于历史数据库的近邻搜索算法为用户推荐同类任务的较好配置;然而,该方法在如何准确地定义"同类型任务"以及过度依赖历史库中历史作业的参数方面存在缺陷,并没有高效、完全地实现自动调优。王国路等[33]将参数优化问题转化为一个机器学习领域的分类问题,从而提出了一种二分类加多分类的多模型融合方法来实现参数的自动调优;但是,该方法只是依据用户经验从 180 个配置参数中选择 13 个作为参数组合,针对不同应用选择决定性参数的准确性和针对性尚未提出更好的解决方案。

4.4　调度优化

在进行 Spark 调度优化分析与研究之前,有必要对 Hadoop MapReduce 与 Spark 之间的关系做一个详细的说明。

Hadoop 是一个由 Apache 基金会开发的分布式系统的基础架构,用户可以在不了解分布式底层细节的情况下开发分布式程序,充分利用集群的优势进行运算和存储。Hadoop 实现了一个分布式文件系统(Hadoop Distributed File System,HDFS)。Hadoop 框架中最核心的设计就是 HDFS 和 MapReduce。HDFS 为海量数据提供了存储,而 MapReduce 为海量数据提供了计算模型。MapReduce 是由 Google 提出的一个处理大量半结构化数据集合的编程模型。不可否认,Hadoop 自 2007 年被推出后就迅速成为了工业界处理大数据的主流技术和系统平台,并且在工业界和学术界都得到了进一步的开发和改进。但是,Hadoop 依然存在一些局限和不足:1)抽象层次低,需要手工编写代码来完成;2)只提供了 Map 和 Reduce 两个操作,表达力欠缺;3)中间结果也放置在 HDFS 文件系统中,时延高,不适用于交互式数据和实时数据的处理;4)迭代式数据处理的性能较差等。为了改进 Hadoop 的不足,Spark 应运而生。Spark 解决了 Hadoop 中存在的一些问题:Spark 提出基于 RDD 的抽象,适合于实时数据处理;Spark 提供很多转换和动作,很多基本操作(如 Join,GroupBy)已经在 RDD 转换和动作中实现;中间结果存放在内存中,内存空间不足时才会写入磁盘;通过在内存中缓存数据,能提高迭代式计算的性能等。

从以上看出,Spark 与 Hadoop 虽有不同之处,但从本质上讲,Spark 扩展了 Hadoop 中核心的 MapReduce 计算模型,使其能高效地支持更多的计算模型,包括交互式查询和流处理。Spark 与 Hadoop MapReduce 都是进行分布式计算,在进行计算时,核心的数据流处理模型并没有改变,只是计算过程中中间结果存放位置的不同(内存和磁盘)导致了计算速度的差异。目前,相比于直接基于 Spark DAG 计算模型的调度优化,基于 Hadoop 的 MapReduce 计算模型的调度优化的相关文献更加丰富,但国内外针对 Spark 调度优化的文献还较少。对于数据流处理模型,若将基于 Hadoop MapReduce 的调度

优化方式进行基于 Spark 特性的相似性改造，将改造后的优化方式应用于 Spark 平台调度优化方面，很大可能上会取得不错的改进效果，因此，Hadoop MapReduce 计算框架的调度优化方式对于 Spark 调度优化有很大的借鉴价值。鉴于这一特性，拟从一个更高的层次，即分布式计算框架的角度，对 Spark 调度优化做进一步总结与分析。

文献[34]总结并对比分析了 9 种国外研究机构针对MapReduce 计算架构的调度优化方法。其中，ARIA（Automatic Resource Inference and Allocation）架构由文献[35]提出，通过 3 个模块（Job 分析、性能模型的优化和调度）相互协作，能提供一种合适的资源分配方案。在优化和调度模块，ARIA 通过 Job 分析器、分析数据库、Slot 估计器、Slot 分配器和 SLO-Scheduler 5 个相互影响的组成模块，解决了系统利用率低和公平性的问题。其中，SLO-Scheduler 是整个调度优化的核心步骤，该调度器根据 EDF 策略定义了所有 Job 的执行顺序，并根据 Job 完成的时长分配了 Job 所需的资源。Delay 调度器由文献[36]提出，尝试解决数据本地化和 Slot 粘滞问题。该方案通过降低公平性来保证 MapReduce 算法中的数据本地性。Dynamic Priority（DP）调度方案[37]实现了一个内嵌于 Hadoop JobTacker 服务的调度插件，该插件可以替代默认的 FIFO 调度器。DP 调度器的核心思想是将调度分为动态优先分配器（Dynamic Priority Allocator）和优先执行器（Priority Enforcer）两部分。前者实现了一组 API 来管理用户预算，用户如果想增加或减少已经分配的资源，则可以改变之前的方案，即工作执行成本取决于市场需求；后者负责控制和实施资源共享。DP 调度器通过一种强制剥夺资源的方式在一定程度上解决了饥饿问题，并且限制了一个节点上每一个 Task 的运行时间，但是没有解决数据的本地化问题。Deadline Constraint（DC）调度器是一个由文献[38]提出的原型，把 Job 的最后期限（deadline）作为输入的一部分，在已提出的 Job 执行成本模型的基础上定义了 Job 的调度策略。该方案有两个限制：1）Job 必须满足其最后期限；2）Job profile 对于 Job 的执行必须是已知的。不能满足上述约束条件的 Jobs 将不会被调度。文献[39]提出了一种自适应调度器（Adaptive Scheduler），针对多任务的 MapReduce 工作负载实现一个调度框架，该框架可以动态地为运行的负载创建性能模型，然后根据模型提供的信息进行调度。动态分享调度器（Dynamic Share Scheduler）由文献[40]提出，该调度器设计了一个单独的 Master 和多个 Worker nodes。当用户提交 Job 后，Master 会分析其执行时间和整个 Job 的松弛度，同时 Master 会监控系统资源以找到超载节点这类不理想节点。最后，Master 执行分区分配算法，决定如何为可用的 Reduce 任务分配已经生成的分区。

调度对应用性能的影响巨大，当我们考虑调度策略时，经常从表 3 所列的 6 个问题的角度来评估该调度策略。下面首先对调度需要解决的 6 个问题进行简要介绍。

1）数据本地化。调度需要选择合适作业的合适任务，任务调度时将数据本地化能减小网络开销，提高作业的执行效率。

2）Slot 粘滞。Slot 粘滞指一个 Job 的 Task 始终在同一个 Slot 执行，即使它的本地化情况不是最优的，仍旧放弃寻找更优的 Slot。

3）倾斜问题。多数 Task 的执行速度较快，少数 Task 的执行时间非常长，或者等待很长时间后提示内存不足，执行失败。

4）饿死问题。饿死状态发生在一个 Task，因为不停地有另一个优先级更高的 Job 出现。当采用不公平调度时，经常出现该问题。

5）利用率。当系统资源没有被用户很好地分配时，将造成利用率不高的问题。

6）公平性。当没有合适的调度机制保证各个用户的优先性时，应当要考虑该问题。

文献[34]总结了上述基于 Task 级别提出的调度优化方式，如表 3 所列，除了上述 6 个问题，其中还包括代表该调度算法是否已发布实现版本或者处于设计原型阶段的已发布/原型。平台指针对 Hadoop 或 Spark 平台设计。以上所有的研究文献均是基于同构集群的理想状态，但在实际的工业环境中经常会遇到个别节点硬件更换或升级等状况，默认的 Spark 任务调度方式没有考虑集群的异构性[41-42]以及节点当前的资源剩余情况。因此，在异构 Spark 集群条件下考虑调度策略的优化也是很有必要的。文献[43]提出了一种基于异构 Spark 集群的自适应任务调度策略 HSATS，该策略通过检测节点的负载及资源利用率，分析检测得到的参数，自适应动态调节节点任务分配的权值，缩短了用户作业的完成时间，提高了异构集群下的资源利用率与服务质量。

表 3 调度优化方式的对比

Table 3 Comparison of scheduling optimization methods

	数据本地化	Slot 粘滞	倾斜	饿死	利用率	公平性	已发布/原型	平台
FIFO				√			R	H/S
FAIR					√	√	R	H/S
Delay		√		√			R	H/S
Capacity					√	√	R	H
DP				√	√		R	H
DC							P	H
BA					√	√	P	H
Data aff..	√	√			√	√	P	H
Hw aff..					√		P	H
DynamicShare			√	√	√	√	P	H

4.5 Shuffle 过程优化

Shuffle 的中文意思是洗牌。之所以需要 Shuffle，是因为具有某种共同特征的一类数据需要最终汇聚到一个计算节点上进行计算。这些数据分布在各个存储节点上，并且由不同节点的计算单元处理。简单来说，将数据重新打乱然后汇聚到不同节点的过程就是 Shuffle。Shuffle 在执行过程中会遇到多种问题：1）Shuffle 过程处理的数据量会很大，需要将 TB 级甚至 PB 级的数据分散到几百甚至数千、数万台机器上；2）为了将数据汇聚到正确的节点位置，需要保证数据被存放在正确的 Partition 中，由于数据大小大于节点的内存，因此在该阶段会有大量的数据从硬盘读到内存中，这个过程中会发生多次硬盘读写；3）数据在传输的过程中，为了提高效率，多数情况下会对数据进行压缩处理，但数据解压带来的时间

消耗又可能抵消由数据压缩带来的优势,在某些情况下可能出现不采用压缩算法反而更优的现象,因此如何在压缩率和解压时间之间进行适当调整也是难点之一;4)数据在 Shuffle 过程中需要通过网络进行传输,数据的序列化和反序列化也会影响 Spark 作业的性能。因此,针对以上问题,本节从 Shuffle 过程中发生数据倾斜时的解决方案、Shuffle 压缩算法的优化、Shuffle 内存优化 3 个方面对目前针对 Shuffle 过程的性能优化方案进行总结和分析。

4.5.1　数据倾斜的解决方案

Shuffle 过程中,经常会发生数据倾斜问题。数据倾斜是指作业在执行过程中,大部分节点上的 Task 执行完毕,但是有一个或者几个节点上的 Task 运行得很慢,导致整个 Spark 作业的执行时间变长。因此,数据倾斜的避免与优化是十分必要的。

解决数据倾斜问题的关键在于找到数据倾斜发生在哪个执行了 Shuffle 操作并且导致了数据倾斜的 RDD 或 Hive 表,并查看其中 key 的分布情况。针对不同的 key 分布与不同的 Shuffle 算子组合情况,有不同的解决方案。美国技术团队[17]总结了 8 种常见的解决方案,如表 4 所列。

表 4　数据倾斜的解决方案

Table 4　Solutions of data skew

适用场景	解决方案	实现原理
Hive 表中 key 分布不均	使用 Hive ETL 预处理数据	通过对 Hive 进行数据预处理,使得 Spark 作业中不需要再使用原来的 Shuffle 算子
少数几个 key 对应大量数据,其余 key 分布均匀	过滤少数导致数据倾斜的 key	过滤掉对作业执行和计算结果不重要的少数几个大数据量的 key 后,这些 key 将不会再参与计算
Shuffle read Task 的默认值太小(默认 200)	提高 Shuffle 操作的并行度	增加 Shuffle read Task 的数量,使原本分配给一个 Task 的多个 key 分配给多个 Task
对 RDD 执行 reduceBy-Key 等聚合类算子操作或 Spark SQL 中使用 group by 进行分组聚合	两阶段聚合(局部聚合+全局聚合)	将原本相同的 key 通过附加随机前缀的方式变成多个不同的 key,再进行局部聚合;接着去掉随机前缀,再进行全局聚合
RDD 或 Spark SQL 中使用 join,且 join 操作中一个 RDD 或表的数据量较小(几百兆或 1～2GB)	将 reduce join 转为 map join	将小 RDD 或小表进行数据广播并通过加上 map 算子来实现与 join 相同的效果
两个同样大数据量 RDD/Hive 表进行 join,其中一个 RDD/Hive 表中少数几个 key 的数据量大,另一个 key 分布均匀	采样倾斜 key 并拆分 join 操作	对于 join 导致的数据倾斜,如果只是某几个 key 导致了倾斜,可将少数几个 key 分拆成独立 RDD,添加随机前缀进行 join
RDD 中大量的 key 因 join 操作出现倾斜	随机前缀和扩容 RDD 进行 join	对与原来一样的 key 随机附加前缀使其变成不一样的 key,处理后的"不同 key"分散到多个 Task 中处理,而不是让一个 Task 处理大量相同的 key;此方法对内存要求较高

从表 4 中可以看到,针对不同的场景,有多种解决方案,但每种解决方案均有其优势和劣势。例如,使用 Hive ETL 预处理数据的方案虽然实现起来较为便捷,但是治标不治本,当数据量较大时,在 Hive ETL 中还是会发生数据倾斜。同样地,在将 reduce join 转为 map join 的方案中,因为只适用于特定场景,广播小表或小 RDD 中的数据时会消耗内存资源,有可能发生内存溢出。

表 4 只是从开发编程方面进行调优,在实践中往往会遇到各种场景混杂的情况,要求开发人员结合自身的开发经验选择不同的方案进行组合,反复实验,力求做到性能最优。

4.5.2　Shuffle 压缩算法的优化

Spark Shuffle 数据压缩所采用的算法有 3 种,分别是 Snappy,LZ4,LZF。3 种不同的压缩算法针对不同的数据流和应用有不同的压缩效果,其性能对比如表 5 所列。由此可见,由于压缩和解压缩的效率不同,不同的压缩算法会对不同的 Spark 应用程序产生极大的影响。Spark 允许用户压缩 Shuffle 过程的中间结果或者输出,或者对两者都进行压缩。Spark 会自动检查输入文件是否为压缩格式,并在需要时进行解压缩。两个相关配置参数 spark.shuffle.compress 和 spark.shuffle.spill.compress 的默认配置都是 true,用来设置 Shuffle 过程中是否对 Shuffle 数据进行压缩。其中,前者针对最终写入本地文件系统的输出文件;后者针对在处理过程中需要写入到外部存储的中间数据,即针对最终的 Shuffle 输出文件。文献[44]利用集成监控组件 Ganglia 的 Spark,监控不同配置参数、不同负载(WordCount,PageRank)、不同数据类型(String 类型、Int 类型)、不同压缩规模(5～25 GB)下不同压缩算法对 Shuffle 过程的性能影响。实验结果表明,对于大部分的应用场景和数据类型,LZ4 算法都是最优的。但是,该研究并没有给出一个可选择的压缩配置,所有结论均通过样本实验得出,并没有为用户提供一个可以在应用程序运行前进行最优参数配置的压缩策略模型。

表 5　不同压缩算法的性能对比

Table 5　Comparison of performance of different compression algorithms

压缩算法	原始文件大小/GB	压缩后文件大小/GB	压缩速度/(MB/s)	解压缩速度/(MB/s)
Snappy	10	2.22	242	683
LZ4	10	2.08	319	1070
LZF	10	2.08	320	482

4.5.3　Shuffle 内存优化

除了对 Shuffle 相关的参数进行合理配置外,对 Shuffle 阶段内部性能的提升也是目前的研究热点。文献[45]提出对 Spark Shuffle 性能进行优化有两种方式:1)合并 Shuffle 过程的中间结果,以提高 Spark 性能;2)创建大的 Shuffle 文件,由于 Spark 的压缩性很低,并且对 Map 端元数据分裂成列及 Reduce 端的重构过程需要大量的计算和数据维护,因此,该优化使用柱状压缩来缓解 CPU 瓶颈,提高 Spark 的运行效率。文献[46]针对 Shuffle file 的性能提出了优化方法,即针对 Shuffle 过程中产生的中间文件,利用列压缩的方式提高数据传输效率,进而提高 Spark 的执行效率。但是这种方式也有不可避免的缺点,即缺乏对迭代的支持,并且将中间数据保存到磁盘上,从而导致延迟较大。该文献对比了早期 Spark 版本中,Shuffle 的原理和目前版本对 Shuffle 实现的改进。早期的 Spark 版本中 ShuffleMapTask 为每个 ShuffleReduceTask 创建一个用于缓存 RDD 计算结果的 ArrayBuffer,Map 计算结束后,缓存中的数据才会通过 BlockManager 写到

磁盘上，ShuffleMapTask 需要一次性存储所有的计算结果，当内存不足时，可能会出现溢出错误。为了解决这个问题，Spark0.8[47]引入 ShuffleBlockManager 类为 Shuffle Write 分配内存并管理磁盘 I/O；同时，M 个 ShuffleMapTask 产生 $M*R$ 个输出文件，其中 R 是 ShuffleReduceTask 的个数。Spark 中 M 和 R 一般都很大，会造成 Shuffle File 过多的问题。根据以上问题，针对 Spark Shuffle 的原有公平分配内存调度算法的不足，文献[48]提出了一种基于溢出历史的自适应内存调度算法。实验表明，该调度算法可以大大提高对数据分布不均匀的应用的处理效率；但是该算法过于依赖溢出历史信息，在存在溢出的前提下可能需要反复调优。目前，针对 Shuffle 过程进行内存优化的重点在于 Shuffle file 如何有效地利用内存和内存调度两方面，可以在参考经典调度机制的前提下结合不同调度算法的优势以及 Spark Shuffle 的工作机制来设计更高效的 Shuffle 内存调度算法。

5 进一步的研究方向

从开发原则的角度来讲，目前针对 Spark 应用开发各个领域的不同业务会有不同的规范，因此我们只需从 4.1 节所总结的基本开发原则入手，针对不同的业务场景，根据侧重优化的内容制定对应的开发原则即可，尽量做到选择合理的数据结构和算子组合。

从内存优化的角度来讲，目前的优化方式存在节点之间和任务之间无法实现内存共享、节点之间的缓存数据及缓存能力不能共享等问题，这会在数据分布不均时浪费缓存能力。从其他节点上获得缓存数据需要借助网络传输，但随着硬件设备的不断发展，网络的速度越来越快，万兆网已经被越来越多的人使用，带宽也在不断提升，这使得网络的传输速度与内存的读写速度之间的差距逐渐变小。在不久的将来，网络传输速度造成的速度损耗可能并不会影响内存优化的速度优势。虽然网络传输速度比内存优化速度慢，但是从磁盘读入或者重新计算都比远程网络缓存的代价高；并且当用户本地缓存不足时，远程网络缓存会变为最佳选择。因此，目前在能判断出读写与磁盘、冲洗计算和远程缓存的代价关系时，运用远程缓存可以进一步提升运行效率；同时，在实际工业应用环境下，机器异构的情况不可避免，如何使 Spark 在异构环境下分配任务时自适应地考虑到不同机器的性能，并根据机器的不同性能分配不同数据量的任务，达到充分利用资源、提高工作效率的目标，也是需要优化的内容。此外，多任务之间的内存共享也是未来研究的重点。

从配置参数优化的角度来看，面向领域的系统运行维护已经成为大数据技术的成本瓶颈。基于系统运行数据，开发者可以获得关于系统的众多信息，研究如何运用关联分析、机器学习、数据挖掘、信息可视化等技术，从系统运行数据信息中挖掘有用信息后做出优化决策变得越来越重要。由于针对 Spark 参数配置优化的研究较少，同时基于 Hadoop MapReduce 的参数配置优化的文献较丰富，鉴于二者在参数配置方面的相似性，可以尝试将后者的思想运用于 Spark 平台，如文献[49-50]基于机器学习的方法，尝试建立高效的性能模型，用于实现 Hadoop MapReduce 配置参数的自动调优。同理，我们可以尝试运用机器学习的方法进行 Spark 参数优化研究，以更精准地提取出核心影响参数，并更快地找到最优参数配置组合。

从调度的角度来讲，目前针对 Spark 调度机制的研究较少，但是基于 MapReduce 调度机制的研究成果却比较丰富，基于分布式框架的角度，可以提取基于 MapReduce 平台的优化方法，尝试将其改进后运用于 Spark 平台，对 Spark 原有的 FIFO 及 FAIR 调度算法进行改进。此外，现有的 Spark 调度算法存在无法满足用户硬实时调度场景的需求，笔者所在实验室正在尝试设计一种综合考虑应用截止期和价值密度的调度算法——DVDA(Deadline and Value Density-Aware)。利用该算法，在 Spark on YARN 的部署模式下实现了一种支持硬实时调度的 DVDAS(Deadline and Value Density-Aware Scheduler)调度器，实验表明该调度器可以明显提高应用的执行成功率。

从 Shuffle 过程的角度看，追溯到 Spark0.6 和 Spark0.7 时，Shuffle 的结果都要先存储到内存中，因此在数据量极大的情况下极有可能发生 GC 和 OOM。Spark0.8 时，Shuffle 的结果改为直接写入磁盘，并且为下游的 Task 都生成一个单独的文件，这样虽然将结果都存入了内存，但是又由于生成的小文件过多造成了随机读取的问题，严重影响了性能。之后，Spark 又引入了 File Consolidation 机制，这在一定程度上解决了该问题。因此，为了不局限于 Hash Based Shuffle 机制，Spark1.1 引进了 Sort Based Shuffle，且目前该机制已成为了 Shuffle 的默认选项。通过了解 Shuffle 机制的源码，用户可以使用 Spark Pluggable 框架开发适用于自己 Spark 业务的 Shuffle 机制，以达到性能的最优化。同时，对 Shuffle 内存调度只有单一的 FAIR(公平)方式，无法适用于单一工作节点上由于数据处理量大造成作业不能分配到充分资源而一直处于等待状态的情况。我们可以尝试对 Task 的溢出次数以及等待时间进行分析，结合最高响应调度方式和最高优先权调度方式进行优化，尝试设计出一种综合考虑长短作业都可以在最短的时间内充分利用资源进行计算的方案，以缩短 Spark 任务的执行时间。此外，在 Shuffle 运行过程中，针对不同任务的不同过程，我们可以设计一个合理的压缩策略算法，使得用户可以在任务运行前获得有效的最佳压缩配置参数，为用户节省重复调优的时间。最后，随着内存成本的不断下降和容量的扩大，在未来是否存在将需要处理的信息全部存储在内存中的情况也是值得考虑的问题。

结束语 随着大数据处理计算框架的不断出现，Spark 在性能和对多种计算模型的支持上都具有突出优势，已经逐渐得到了广泛应用。随着 Spark 在国内外的广泛应用，其性能优化成为了需要解决的核心问题。Spark 大数据平台底层执行机制的复杂性，使得普通用户很难找到性能瓶颈，更无从对 Spark 性能进行优化。本文主要介绍了 Spark 的工作原理，并从开发原则、内存、配置参数、调度、Shuffle 过程 5 个方面提出了解决 Spark 性能优化问题的基本思路，以期为用户优化 Spark 性能、进一步提高 Spark 集群的资源利用率提供参考。最后，结合 Spark 不同工作过程的不同特性，探讨了未来进一步的研究方向。

参 考 文 献

[1] ZAHARIA M. Anarchitecture for fast and general data processing on large clusters[M]. Morgan & Claypool,2016.

[2] 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.

[3] 高彦杰. Spark 大数据处理:技术,应用与性能优化[M]. 北京:机械工业出版社,2015.

[4] ApacheSpark[EB/OL]. [2017-3-15]. http://Spark. apache. org.

[5] ApacheHadoop[EB/OL]. [2017-3-20]. http://apache. hadoop. org.

[6] Apache Mesos[EB/OL]. [2017-4-18]. http://mesos. apache. org.

[7] Apache Hbase[EB/OL]. [2017-4-18]. http://hbase. apache. org.

[8] ApacheCassandra[EB/OL]. [2017-4-23]. https://cassandra. apache. org.

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

[10] Apache Pig[EB/OL]. [2017-4-25]. http://pig. apache. org.

[11] ApacheHive[EB/OL]. [2017-4-25]. https://hive. apache. org.

[12] BU Y,HOWE B,BALAZINSKA M,et al. HaLoop:efficient iterative data processing on large clusters[J]. Proceedings of the VLDB Endowment,2010,3(1/2):285-296.

[13] BU Y,HOWE B,BALAZINSKA M,et al. The HaLoop approach to large-scale iterative data analysis[J]. The VLDB Journal—The International Journal on Very Large Data Bases,2012,21(2):169-190.

[14] ANANTHANARAYANAN G,GHODSI A,WANG A,et al. PACMan:coordinated memory caching for parallel jobs[C]// Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association,2012:20.

[15] PAVLO A,PAULSON E,RASIN A,et al. A comparison of approaches to large-scale data analysis[C]// Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM,2009:165-178.

[16] JIANG D,OOI B C,SHI L,et al. The performance of MapReduce:an in-depth study[J]. Proceedings of the VLDB Endowment,2010,3(1/2):472-483.

[17] LI X R. Meituan Comment Techical Group. Spark Performance Tuning Guide[EB/OL]. [2017-04-28]. http://tech. meituan. com/Spark-tuning-basic. html.

[18] PIAO H Q,CHEN Y G,DU X Y,et al. Equi-join optimization on Spark[J]. Journal of East China Normal University(Natural Science),2014(5):261-270. (in Chinese)
卞昊穹,陈跃国,杜小勇,等. Spark 上的等值连接优化[J]. 华东师范大学学报(自然科学版),2014(5):261-270.

[19] BLANAS S,PATEL J M,ERCEGOVAC V,et al. A comparison of join algorithms for log processing in mapreduce[C]// Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM,2010:975-986.

[20] SAKR S,LIU A,FAYOUMI A G. The family of mapreduce and large-scale data processing systems[J]. ACM Computing Surveys (CSUR),2013,46(1):11.

[21] CHEN K,WANG B,FENG L. Data Object Cache in Spark Computing Engine[J]. ZTE Technology Journal,2016,22(2):23-27. (in Chinese)
陈康,王彬,冯琳. Spark 计算引擎的数据对象缓存[J]. 中兴通讯技术,2016,22(2):23-27.

[22] FENG L. Research and Implementation of Memory Optimazation Based on Parallel Computing Engine Spark[D]. Beijing:Tsinghua University,2013. (in Chinese)
冯琳. 集群计算引擎 Spark 中的内存优化研究与实现[D]. 北京:清华大学,2013.

[23] CHURILA S A,ZHOU G L,SHI L,et al. Parallel cube computing in Spark[J]. Journal of Computer Applications,2016,36(2):348-352. (in Chinese)
萨初日拉,周国亮,时磊,等. Spark 环境下并行立方体计算方法[J]. 计算机应用,2016,36(2):348-352.

[24] LI M,TAN J,WANG Y,et al. Sparkbench:a comprehensive benchmarking suite for in memory data analytic platform spark[C]// Proceedings of the 12th ACM International Conference on Computing Frontiers. ACM,2015:53.

[25] HERODOTOU H,LIM H,LUO G,et al. Starfish:A Self-tuning System for Big Data Analytics[C]// Fifth Biennial Conference on Innovative Data Systems Research, Asilomar. DBLP,2011:261-272.

[26] HERODOTOU H,BABU S. Profiling,what-if analysis,and cost-based optimization of mapreduce programs[J]. Proceedings of the VLDB Endowment,2011,4(11):1111-1122.

[27] HERODOTOU H. Hadoop performance models[J]. arXiv preprint arXiv. 2011,1106. 0940.

[28] WU D,GOKHALE A. A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration[C]// 20th Annual International Conference on High Performance Computing. IEEE,2013:89-98.

[29] WU D. A Profiling and Performance Analysis based Self-tuning System for Optimization of Hadoop MapReduce Cluster Configuration[D]. Nashvile:Vanderbilt University,2013.

[30] CHEN C O,ZHUO Y Q,YEH C C,et al. Machine Learning-Based Configuration Parameter Tuning on Hadoop System[C]// 2015 IEEE International Congress on Big Data (BigData Congress). IEEE,2015:386-392.

[31] RAVI N. Configuring and optimizing Spark applications with ease-Nishkam ravi,Cloudera[EB/OL]. (2015-09-01). https://apachebigdata2015. sched. org/event/55afa6d65370a56bdbcb5eba5166f010#. VemuzvaqpEN.

[32] CHEN Q A,LI F,CAO Y,et al. Parameter optimation for Spark jobs based on runtime data analysis[J]. Computer Engineering & Science,2016,38(1):11-19. (in Chinese)
陈侨安,李峰,曹越,等. 基于运行数据分析的 Spark 任务参数优化[J]. 计算机工程与科学,2016,38(1):11-19.

[33] XU J G,WANG G L,LIU S Y,et al. A Novel Performance Evaluation and Optimization Model for Big Data System [C]// Proceedings of the 15th International Symposium on Parallel and Distributed Computing (ISPDC 2016). Fuzhou, China, 2016:1765-1773.

(下转第 37 页)

panion. 2017：1477-1483.

[7] HAN H，WANG W Y，MAO B H. Borderline-SMOTE：a new over-sampling method in imbalanced data sets learning[C]// International Conference on Intelligent Computing. Berlin，Springer，Heidelberg，2005：878-887.

[8] CIESLAK D A，CHAWLA N V，STRIEGEL A. Combating imbalance in network intrusion datasets[C]// IEEE International Conference on Granular Computing. IEEE，2006：732-737.

[9] LI M，FAN S. CURE-SMOTE algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests[J]. Bmc Bioinformatics，2017，18(1)：169.

[10] LI J，FONG S，SUNG Y，et al. Adaptive swarm cluster-based dynamic multi-objective synthetic minority oversampling technique algorithm for tackling binary imbalanced datasets in biomedical data classification[J]. Biodata Mining，2016，9(1)：37.

[11] LIU X Y，WU J，ZHOU Z H. Exploratory Undersampling for Class-Imbalance Learning[J]. IEEE Transactions on Systems Man & Cybernetics Part B Cybernetics A Publication of the IEEE Systems Man & Cybernetics Society，2009，39(2)：539-550.

[12] SEIFFERT C，KHOSHGOFTAAR T M，VAN HULSE J，et al. RUSBoost：A hybrid approach to alleviating class imbalance[J]. IEEE Transactions on Systems，Man，and Cybernetics-Part A：Systems and Humans，2010，40(1)：185-197.

[13] RODRÍGUEZ J J，KUNCHEVA L I，ALONSO C J. Rotation forest：A new classifier ensemble method[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence，2006，28(10)：1619-1630.

[14] LIN T Y，GOYAL P，GIRSHICK R，et al. Focal Loss for Dense Object Detection[OL]. http://www. researchgate. net/publication/322059369-Focal-Loss-for-Dense_Object-Detection.

[15] GOODFELLOW I J，POUGET-ABADIE J，MIRZA M，et al. Generative adversarial nets[C]// International Conference on Neural Information Processing Systems. MIT Press，2014：2672-2680.

[16] ARTHUR A，DAVID N. The UCI Machine Learning Repository [DB/OL]. http://archive. ics. uci. edu/ml/datasets. html.

[17] CHEN S，HE H，GARCIA E A. RAMOBoost：Ranked Minority Oversampling in Boosting[J]. IEEE Transactions on Neural Networks，2010，21(10)：1624-1642.

（上接第 15 页）

[34] RUMI G，COLELLA C，ARDAGNA D. Optimization Techniques within the Hadoop Eco-system：A Survey[C]// 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). IEEE，2014：437-444.

[35] VERMA A，CHERKASOVA L，CAMPBELL R H. ARIA：automatic resource inference and allocation for mapreduce environments[C]// Proceedings of the 8th ACM International Conference on Autonomic Computing. ACM，2011：235-244.

[36] SANDHOLM T，LAI K. Dynamic proportional share scheduling in hadoop[C]// Workshop on Job Scheduling Strategies for Parallel Processing. Springer Berlin Heidelberg，2010：110-131.

[37] RAO B T，REDDY L S S. Survey on improved scheduling in Hadoop MapReduce in cloud environments[J]. arXiv preprint arXiv：1207. 0780，2012.

[38] KC K，ANYANWU K. Scheduling hadoop jobs to meet deadlines[C]// IEEE Second International Conference on Cloud Computing Technology and Science. IEEE，2011：388-392.

[39] VERMA A，CHERKASOVA L，KUMAR V S，et al. Deadline-based workload management for mapreduce environments：Pieces of the performance puzzle[C]// Network Operations and Management Symposium (NOMS). IEEE，2012：900-905.

[40] ZACHEILAS N，KALOGERAKI V. Real-Time Scheduling of Skewed MapReduce Jobs in Heterogeneous Environments[C]// ICAC. 2014：189-200.

[41] XU X，CAO L，WANG X. Adaptiv e task scheduling strategy based on dynamic workload adjustment for heterogeneous Hadoop clusters[J]. IEEE Systems Journal，2016，10(2)：471-482.

[42] NIGHTINGALE E B，CHEN P M，FLINN J. Speculative execution in a distributed file system [J]. ACM SIGOPS Operating Systems Review，2005，39(5)：191-205.

[43] YANG Z W，ZHENG Q，WANG S，et al. Adaptive Task Scheduling Strategy for heterogeneous Spark Cluster[J]. Computer Engineering，2016，42(1)：31-35，40. (in Chinese)
杨志伟，郑烇，王嵩，等. 异构 Spark 集群下自适应任务调度策略[J]. 计算机工程，2016，42(1)：31-35，40.

[44] KANG H M. Research on Spark Optimization Based on Fine-Grained Monitoring[D]. Harbin：Harbin Institute of Technology，2016. (in Chinese)
康海蒙. 基于细粒度监控的 Spark 优化研究[D]. 哈尔滨：哈尔滨工业大学，2016.

[45] RANA N，DESHMUKH S. Shuffle Performance in Apache Spark[C]// International Journal of Engineering Research and Technology. ESRSA Publications，2015.

[46] DAVIDSON A，OR A. Optimizing Shuffle performance in Spark [R]. University of California，Berkeley-Department of Electrical Engineering and Computer Sciences，2013.

[47] JASON D. Consolidating Shuffle Files in Spark[EB/OL]. [2017-04-28]. https://issues. apache. org/jira/browse/SPARK-751.

[48] CHERN Y Z. Analysis and optimization of Memory Scheduling Algorithm of Spark Shuffle[D]. Hangzhou：Zhejiang University，2016. (in Chinese)
陈英芝. Spark Shuffle 的内存调度算法分析及优化[D]. 杭州：浙江大学，2016.

[49] YIGITBASI N，WILLKE T L，LIAO G，et al. Towards machine learning-based auto-tuning of mapreduce[C]// 2013 IEEE 21st International Symposium on Modelling，Analysis and Simulation of Computer and Telecommunication Systems. IEEE，2013：11-20.

[50] CHEN C O，ZHUO Y Q，YEH C C，et al. Machine Learning-Based Configuration Parameter Tuning on Hadoop System[C]// 2015 IEEE International Congress on Big Data. IEEE，2015：386-392.