**海量存储论文**

面向固态硬盘的Spark数据持久化方法设计

****

院（系）名 称 ：计算机系

学 生 姓 名 ：张天泽

学 号 ：2017202110050

二零一七年十二月

面向固态硬盘的Spark数据持久化方法设计

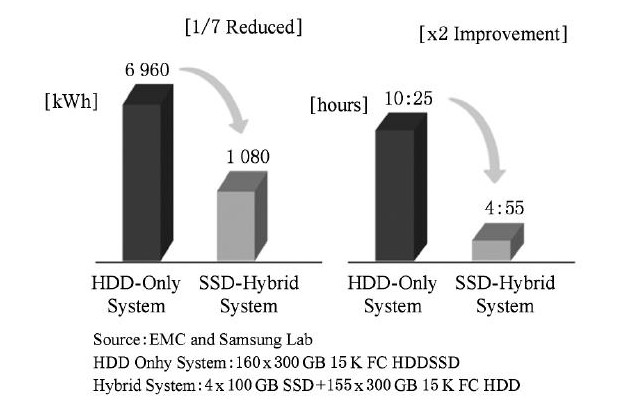
**摘要**：基于固态硬盘(solid-state drive,SSD)和硬盘(hard disk drive,HDD)混合存储的数据中心已经成为大数据计算领域的高性能载体,数据中心负载应该可将不同特性的数据按需持久化到SSD或HDD,以提升系统整体性能.Spark是目前产业界广泛使用的高效大数据计算框架,尤其适用于多次迭代计算的应用领域,其原因在于Spark可以将中间数据持久化在内存或硬盘中,且持久化数据到硬盘打破了内存容量不足对数据集规模的限制.然而,当前的Spark实现并未专门提供显式的面向SSD的持久化接口,尽管可根据配置信息将数据按比例分布到不同的存储介质中,但是用户无法根据数据特征按需指定RDD的持久化存储介质,针对性和灵活性不足.这不仅成为进一步提升Spark性能的瓶颈,而且严重影响了混合存储系统性能的发挥.有鉴于此,首次提出面向SSD的数据持久化策略.探索了Spark数据持久化原理,基于混合存储系统优化了Spark的持久化架构,最终通过提供特定的持久化API实现用户可显式、灵活指定RDD的持久化介质.基于SparkBench的实验结果表明,经本方案优化后的Spark与原生版本相比,其性能平均提升14.02%.

1. **研究背景**

1.1存储异构数据中心的发展

IDC的研究报告指出:“自2005年至2020年, 人类社会所产生的数据将增长300倍———从130艾字节增长到40 000艾字节”.数据的爆炸式增长从根本上改变了传统数据的规模、种类和结构等, 催生了新一代数据中心的数据存储和管理方式的变革.尤其对于大数据分析型数据中心, 其任务高频率的读、写磁盘操作对数据中心存储系统的性能提出巨大的挑战.SSD的引入, 显著地提升了数据中心的性能和能效.SSD可以帮助企业在一个快速发展的大数据时代中最大限度地提升数据的存储和分析效率.目前, 诸多新兴技术可以有效地提升SSD的I/O带宽和降低访问延迟.而HDD仍然能为那些对存储性能要求较低的数据提供大量的存储效率.IDC的研究数据显示, 大量的数据被数据中心收集并捕获后, 并不经常被访问, 称之为冷数据, 约占全球数据的90%;而剩余的10%的数据被收集并捕获后, 会经常性地被访问, 称之为热数据.显然, 将全部的数据都存储在高性能、低延迟的存储设备是不合理的, 成本是极为昂贵的.因此, 将SSD和HDD以合理的方式进行组合, 通过构建混合存储系统有可能带来性能的大幅提升, 同时保障成本可控.

目前, 基于SSD和HDD混合存储的异构数据中心在学术界得到广泛研究, 其从组织结构上分为2类:1) SSD作为HDD的缓存SSD作为HDD的同层持久化存储.在SSD作为HDD的缓存组织结构中, SSD负责存放HDD少量数据的拷贝, 所有的数据请求优先查找SSD, 如果请求的数据在SSD中就直接由SSD服务, 否则从HDD上拷贝相应的数据到SSD中.这种分层结构, 通过部署高命中率的数据映射机制, 可以有效地提升存储系统的I/O性能.在另一种结构中, SSD作为HDD的同层持久化存储, 用户或系统可以根据数据的冷热条件将不同类别的数据按需持久化到SSD或HDD, 利用SSD高速读、写的特征来提升热数据的访问效率, 同时, 利用HDD大容量的特性来提升数据的存储效率, 从而提升系统的整体性能.本文基于该结构展开面向固态硬盘的Spark数据持久化方法的研究.在工业界, 基于SSD和HDD混合存储的异构数据中心已经得到了普遍的应用.如Google、Facebook、百度以及阿里云等国内外大型互联网公司都已经将SSD引入数据中心, 并结合自身业务需求, 合理地构建了基于SSD和HDD混合存储的异构数据中心.如图1所示, 三星公司的研究报告指出, 基于SSD和HDD的混合存储系统可以大幅度地降低数据中心能耗至原来的1/7, 同时将数据中心性能提升2倍

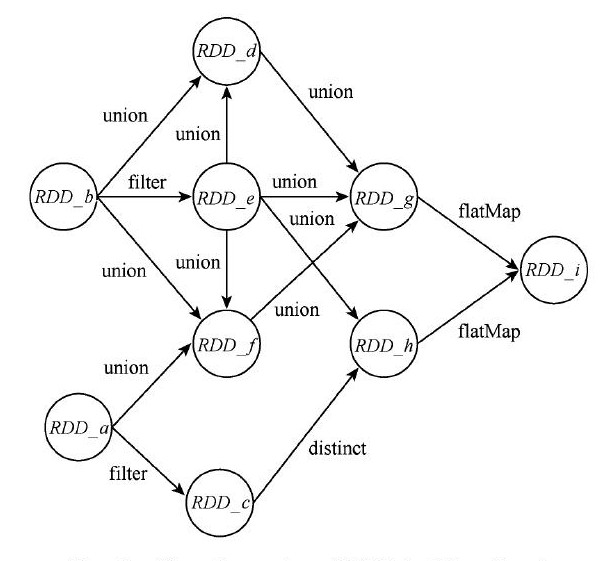


1.2混合存储系统中的Spark数据持久化问题

Spark的功能涵盖了大数据计算的各个分支领域, 如SQL类处理、实时流数据处理 机器学习和图计算等复杂计算任务.Spark广泛的应用范围、简洁易用的API, 使得Spark已经成为越来越受欢迎的大数据计算平台之一.Spark最重要的一项功能是持久化RDD分区数据到内存或硬盘, 被持久化的分区数据可被其他迭代任务直接读取, 避免了重复计算.

在混合存储系统中, 根据数据的冷热条件将不同类别的数据按需持久化到SSD或HDD, 可有效提升数据的访问和存储效率.具体到Spark应用中, 不同RDD的重复利用率往往不同, 换言之, 不同RDD的冷热度存在明显差异.因此, 在仅考虑存储层的持久化问题时, 如将高热度的RDD分区数据完全持久化到SSD, 将热度相对较高或基于其他原因的关键RDD分区数据持久化到HDD, 充分发挥混合存储的特性, 可有效加速大数据的计算速度.然而, Spark所提供的持久化语义对存储介质不具备感知能力, 无法实现RDD分区数据的按需持久化.

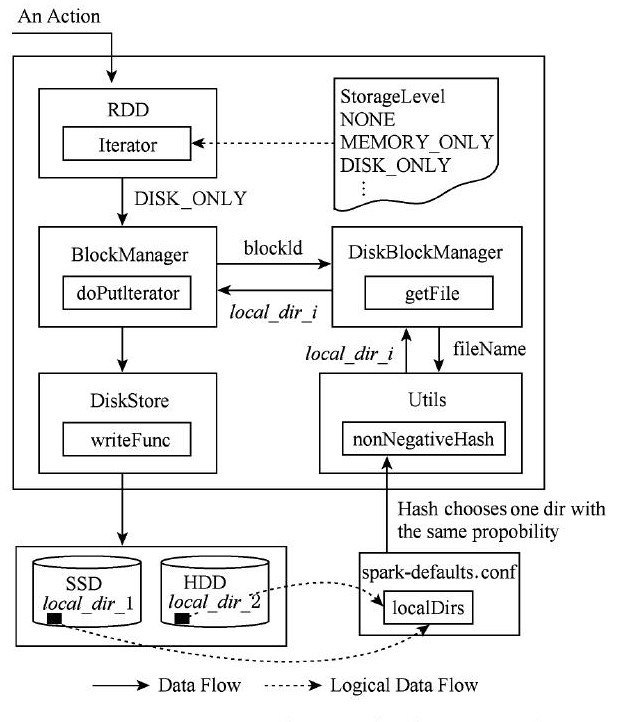
针对该问题, 做3组实验进行验证:将原生Spark2.0部署到仅配置一块HDD、仅配置一块SSD以及同时配置一块HDD和一块SSD三种服务器平台上, 同时在Spark配置文件中, 每一块存储设备配置一个临时文件目录.图2所示为我们设计的MicroBench的RDD转换关系, 其中RDD\_a, RDD\_b和RDD\_e的依赖度比较高, 所以我们对这3个RDD做persist (DISK\_ONLY) 持久化.以RDD的分区为单位, 分别统计3种情况下被持久化的数据分布情况.



统计实验结果发现, RDD\_a, RDD\_b和RDD\_e全部被划分为28个分区.第1组实验中, 3个RDD的分区数据全部被持久化到HDD;第2组实验中, 3个RDD的分区数据全部被持久化到SSD;第3组实验中, 每个RDD的14个分区被持久化到SSD, 另一半被持久化到HDD.

统计结果表明, 被持久化的RDD分区数据的分布和各实验中临时文件目录的配置存在一定关系, 即原生Spark仅按照临时文件目录个数比例来确认持久化数据的存储位置.而在SSD和HDD混合存储系统中, 由于RDD\_e的依赖度较高, 希望将RDD\_e完全持久化到SSD而RDD\_a和RDD\_b持久化到HDD, 以此提升Spark数据处理效率是无法实现的.因此, 原生Spark所提供的持久化语义灵活性较差, 程序员无法根据Spark应用数据的特征, 显式地依据RDD的冷热特性实现按需持久化.有鉴于此, 本文将进一步探索面向SSD的数据持久化方法.

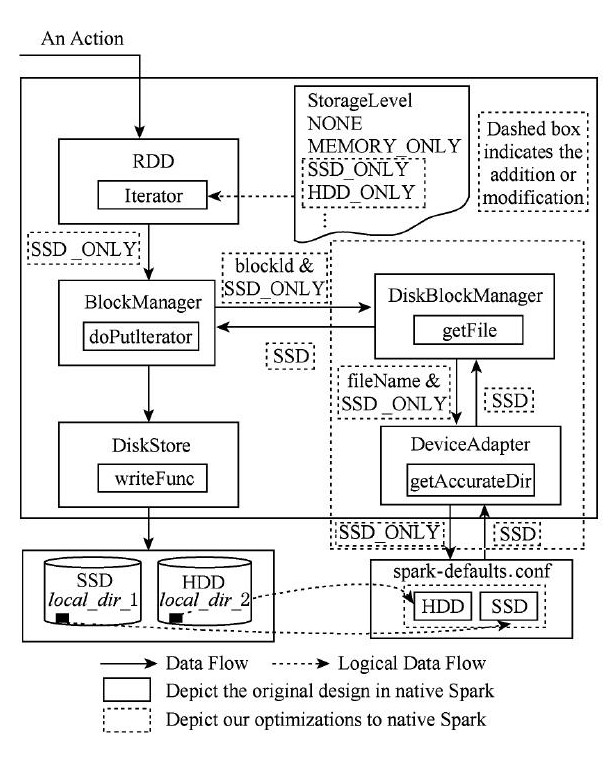
原生Spark持久化框架如图3所示.其中, 临时文件目录由用户通过配置文件配置, 且可以同时配置多个.临时文件目录的选择决定了数据持久化地址, 具体一个RDD的某一分区数据持久化地址的选择是由Utils模块的nonNegativeHash方法完成, 该函数的设计原理是保持每一个目录以相同的概率被选取.如上文所做的第3组实验中, 当配置了2个临时文件目录时, 每个目录都有50%被使用的概率.



1. **面向SSD的持久化**

基于1.2节分析, 研究Spark如何正确地感知混合存储系统底层存储设备, 提供更为灵活的持久化API.首先, 将底层存储设备的差异暴露给用户, 打破DISK\_ONLY的屏蔽作用, 并向用户提供更为精确的持久化API, 实现Spark应用程序的按需持久化, 具体实现如下.

如图4所示, Spark持久化架构的具体优化方案如下: 增加“SSD”和“HDD”临时文件目录管理变量, 将临时文件目录的混合管理方式改为由“SSD”和“HDD”分别管理指向SSD和HDD的临时文件目录; 增加设备适配器DeviceAdapter模块, 接收用户设置的数据持久化级别, 同时读取用户配置的临时文件目录, 实现持久化级别参数到SSD或HDD的精确映射; 增加SSD\_ONLY和HDD\_ONLY两个持久化级别, 将混合存储系统特征暴露给用户.同时, 扩展StorageLevel的作用域, 如图3所示, StorageLevel



仅作用于BlockManager, 为用户和BlockManager提供数据持久化级别.我们将StorageLevel作用域进一步延伸至DeviceAdapter模块. 其中, DeviceAdapter的算法设计如下所述. 算法1.固态硬盘/磁盘持久化地址适配算法. 输入:数据持久化级别StorageLevel、用户标记的级别storageLevel、临时文件目录SSDLocation和HDDLocation; 输出:数据持久化物理地址address.

1. **性能评估**

3.1MicroBench性能评估

本节通过MicroBench性能评估实验, 验证面向固态硬盘的Spark数据持久化方法设计的正确性和灵活性, 即实现了面向固态硬盘的Spark数据持久化方法.按图2所示设计方案, 首先验证优化后Spark可以做到RDD分区数据按需持久化.对RDD\_e调用persist (SSD\_ONLY) 做持久化, 而对RDD\_a和RDD\_b则调用persist (HDD\_ONLY) 做持久化, 统计结果如表4所示.统计结果显示, 优化后Spark的持久化结果完全按照程序员所调用的持久化API对分区数据进行存储, 实现了Spark应用程序中间数据面向固态硬盘的持久化, 进而提升了Spark持久化API的灵活性.

下面分别对优化后的Spark和原生Spark, 变化不同RDD的持久化介质进行更加深入的实验分析.

图5显示的结果为:依次分别针对每一个RDD调用2种不同的持久化方法, 图5所示的MicroBench的执行时间, 其中persist (DISK\_ONLY) 和+persist (SSD\_ONLY) 分别表示原生Spark和提供面向SSD的持久化接口的优化Spark.根据结果可获知, 在支持混合存储系统的平台上, 相比于默认的按比例持久化方案, 显式调用面向SSD的接口进行持久化可以获得更好的性能提升, 这是因为该接口可将RDD的全部分区都存储于SSD中, 而不是原来的按比例在介质间分配分区.同时, 性能提升的幅度也和RDD的访问特征有着密切的关系.例如, 根据图2, RDD\_e的依赖本来较多, 但是由于它是利用filter操作生成的, 使得其规模不大, 因此即使不对其进行持久化性能影响也并不大, 这就是图5中持久化RDD\_e的性能收益并不显著的原因.

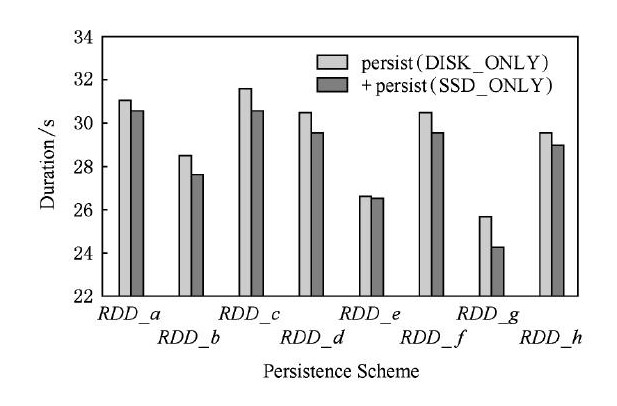
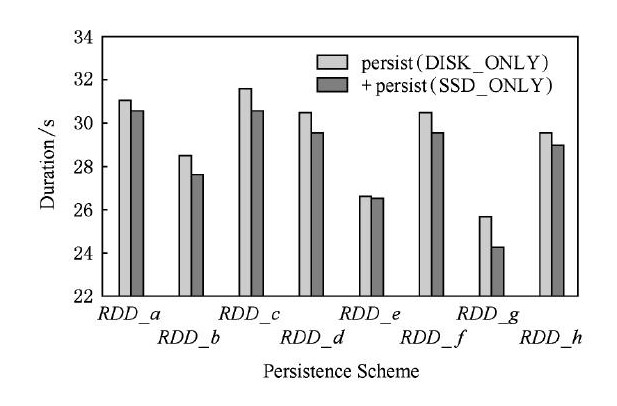


图6是从RDD的串并行关系和RDD的计算复杂度、RDD的热度等多个角度考虑不同持久化方案的比较, 探索将哪些RDD按需持久化到SSD或HDD可以最大限度地提升系统性能.综合图5和图6, 对RDD\_ (c, h) 调用本文提供的接口做持久化性能提升得更加明显, 其原因在于这些RDD的计算更加复杂并且后续使用频繁, 将其全部分区都持久化到SSD可获得更大的性能收益.实验结果表明, 相对于原生Spark, 本文所提持久化方案可以将性能平均提升2.7%, 最大提升4.7%.

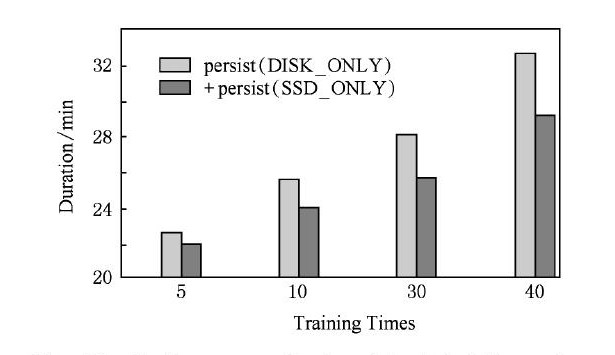


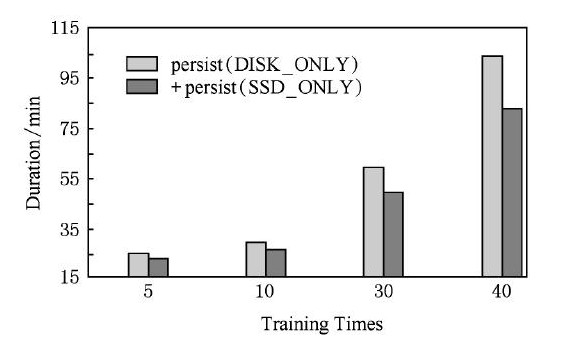
对已经持久化到存储系统层的RDD, 再选择持久化到SSD或HDD, 能够进一步获得的性能提升将与生成该RDD的计算复杂度以及该RDD后续被访问的频繁程度 (热度) 有很大的关系, 不妨将其称之为依赖系数 (RDD的依赖系数为该RDD的子RDD个数) .由此, 将高依赖系数的RDD分区数据持久化至SSD, 相比于系统默认的按分区比例的持久化方案将获得更加明显的性能提升.尽管MicroBench实验结果显示, 优化后Spark性能提升幅度有限, 但这是当前使用的数据规模和应用复杂度简单所导致的, 显然可以构造出效果更佳明显的RDD访问行为.

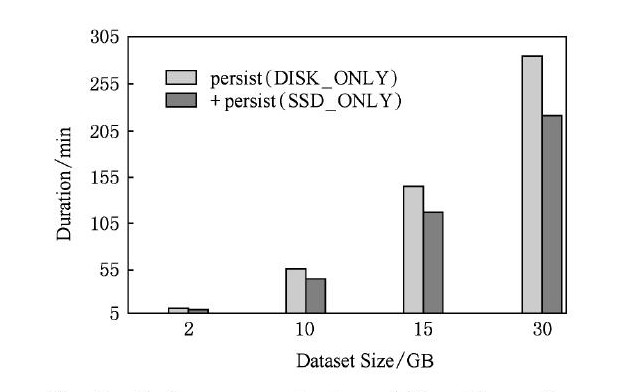
3.2SparkBench性能评估

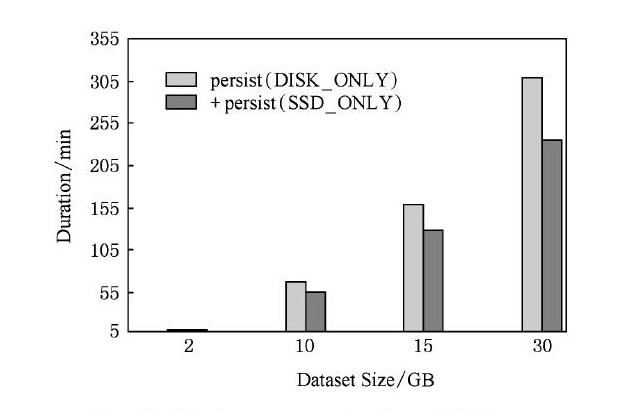
Spark作为目前流行的大数据计算框架, 其十分擅长多次迭代计算任务 (如机器学习、图计算等) .Spark应用程序的迭代次数与RDD持久化性能提升具有密切联系, 显然, 对于迭代次数多的RDD, 将其持久化到SSD对Spark性能提升的帮助更大.另外, 数据规模是Spark读、写硬盘时间消耗的重要影响因素.本节我们基于SparkBench实现了若干机器学习和图计算负载, 从数据量和迭代计算角度比较原生Spark和优化后Spark的性能.同时强调面向固态硬盘的Spark数据持久化方法的设计, 旨在通过向Spark用户提供显示的面向固态硬盘的持久化接口, 提升Spark数据持久化接口的灵活性和针对性, 进而使得程序员可以根据Spark应用程序的数据特征, 实现中间数据的按需持久化.结合SSD, HDD的各自特征与Spark应用程序数据的冷热特性, 将依赖系数较高的RDD持久化到SSD, 以提升中间数据重复利用的读取效率, 将依赖系数相对较低的RDD持久化到HDD, 以减轻SSD因频繁读写而引起擦除操作的压力, 同时扩大了Spark应用程序可利用的存储空间.而具体RDD持久化接口的选择则依赖于程序员对应用程序的个人经验.如下为SparkBench性能评估实验结果的详细分析.

图7和图8分别测试KMeans和Linear-Regression负载性能, 通过改变数据规模评估原生Spark和优化后Spark, 实验结果显示, 系统性能平均提升19.72%, 最大提升20.5%.同样, 针对机器学习算法, 我们测试程序迭代次数对Spark性能的影响, 图9和图10显示结果为不同训练规模下, 原生Spark和优化后Spark的性能比较, 优化后Spark性能平均提升10.04%, 最大提升13.95%.



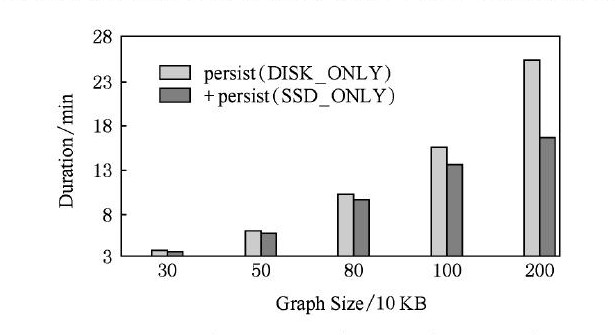


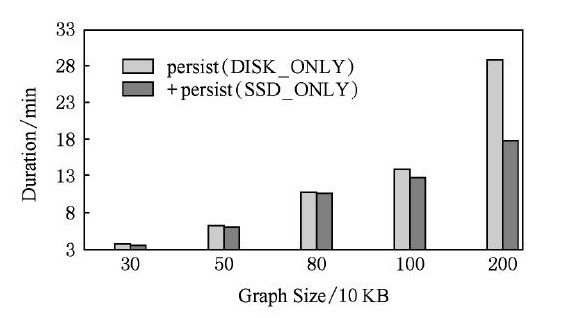




Spark GraphX是一个分布式图处理框架, 它是基于Spark平台提供对图计算和图挖掘简洁易用的而丰富的接口, 极大地方便了对分布式图处理的需求.图11和图12分别通过测试SparkBench的典型图计算负载PageRank和ShortestPath进行评估优化后Spark性能的优越性.随着图规模的不断扩大, 图的结构不断复杂化, 即对Spark性能要求更高, 实验结果表明, 优化后Spark性能平均提高11.79%, 最大提高12.62%, 且随着图的结构越复杂, 优化后Spark的性能越显著.

MicroBench和SparkBench性能评估结果说明, 我们所提出并实现的按需持久化方案的性能明显优越于原生Spark, 该方案实现了Spark用户依据应用程序特征, 对RDD进行按需持久化, 进一步提升了Spark大数据计算性能.经过分析RDD的DAG图可以给出一些使用新的编程接口的启发:对于自身计算复杂度较高并且后续使用比较频繁的RDD, 有针对性地将其持久化至SSD中, 相比于Spark原有的处理方式将获得更大的性能提升, 当然这需要用户对所开发的应用理解得更加深入, 特别是不同RDD的访问行为.





1. **相关工作**

进入“大数据”时代, 大数据种类和数据量的激增以及其结构的复杂化给数据分析带来了巨大的挑战.如何快速、准确地挖掘隐含在大数据内部的高价值信息是目前研究的热门问题, 因此, 人们研发出MapReduce, Spark等大数据计算框架.目前, 国内外关于Spark的研究工作层出不穷, 较多国内文献利用Spark的高性能优势设计了出色的大数据分析算法<a class="sup">[21]</a>.文献[4]介绍了大数据挖掘的若干挑战性问题, 文献[22]回顾了近年来有关大数据计算框架、存储及数据分析等问题的发展.目前较为流行的大数据计算框架有Hadoop和Spark, 其中Spark基于内存计算且能够持久化计算的中间结果, 大幅提升了迭代计算任务的效率.MapReduce是目前已经非常成熟的大数据并行计算框架, 且得到了较为普遍的应用.文献[23-25]分别研究MapReduce应用于快速大规模数据检索、可扩展的云数据匿名化方案和基于MapReduce模型实现了关键字感知的服务推荐系统, 有效地提升了互联网大数据分析工作的效率.另外, 文献[26]提出并设计了Mammoth模型, 该模型通过内存调度算法实现内存的全局管理, 并设计启发式算法来优化执行单元之间的资源分配, 提高了MapReduce性能.

文献[27-29]中, 作者分别从不同的角度出发研究如何提升Spark性能.其中, 文献[28]提出自适应调整策略, Spark在JVM的基础上对内存做了进一步的管理, 作者提出动态地调整Spark内存分配方案, 进一步提升Spark数据处理效率.检查点是Spark实现RDD容错的一个重要概念, 通过将某RDD设置为检查点为快速恢复RDD提供了保障.文献[29]提出了自动检查点设置算法, 以提升RDD的容错性能、改进Spark迭代计算任务的效率, 从而实现提升Spark性能的目的.作者在文献[27]中分析了Spark大数据处理过程中Shuffle的产生及其对Spark性能造成的影响, 并介绍了基于排序的shuffle可以提升Spark性能并应用到Spark1.1.0版本中.伴随着人工智能产业的发展, 机器学习成为当今学术界和产业界共同的热门话题, 且基于机器学习的大数据分析技术得到快速发展.由于Spark是基于内存的计算框架, 且实现了计算中间结果的持久化功能, 使得其在诸如机器学习、图计算等多次迭代计算领域表现出色.作者在文献[30]中介绍了Spark并行分布式机器学习库, 该库基于数据和模型并行策略, 实现对数据和模型存储和相关操作.文献[31]使用Spark更高效地利用大数据实现优秀的推荐系统引擎.

目前, 关于Spark性能提升研究集中于Spark内存管理, 通过调节JVM内存利用率提升Spark性能.然而, 内存空间的不足限制了Spark通过内存加速超大规模数据的计算, 而持久化数据到硬盘打破了内存容量不足对数据集规模的限制, 使得Spark处理大数据游刃有余.本文首次基于混合存储系统提出Spark性能优化方案.通过优化Spark的持久化框架, 实现Spark数据的按需持久化.进而, Spark可以根据程序特征, 将高热度RDD的分区数据持久化到SSD, 利用存储异构数据中心的特征有效地提升Spark性能.

1. **总结**

目前Spark无法精确持久化RDD分区数据到SSD或HDD, 无法根据应用程序特征进行按需持久化, 导致Spark无法充分利用混合存储系统的弹性、高性能的优势以提升自身性能.针对该问题, 本文探索了Spark数据持久化原理, 进而针对混合存储系统对Spark的持久化框架进行优化, 并向用户提供了SSD\_ONLY和HDD\_ONLY持久化API, 实现了Spark的显示SSD和HDD持久化功能.本文通过分析HDD和SSD读取性能并结合RDD读取特征, 理论论证了将RDD分区数据持久化到SSD或HDD以及混合存储系统的性能差异.然后进一步通过基准性能评估实验验证了优化后Spark对应用程序性能提升明显优于原生Spark.未来将进一步探索面向异构存储系统的隐式持久化方案, 即数据持久化介质的确定由Spark框架自动完成, 对用户是透明的.

**参考文献**

1. Labrinidis A, Jagadish H V.Challenges and opportunities with big data[J].Proceedings of the VLDB Endowment, 2012, 5 (12) :2032-2033
2. Howe D, Costanzo M, Fey P, et al.Big data:The future of biocuration[J].Nature, 2008, 455 (7209) :47-50
3. Kriegel H P, Borgwardt K M, Kr9ger P, et al.Future trends in data mining[J].Data Mining and Knowledge Discovery, 2007, 15 (1) :87-97
4. Wu Xindong, Zhu Xingquan, Wu Gongqing, et al.Data mining with big data[J].IEEE Trans on Knowledge and Data Engineering, 2014, 26 (1) :97-107
5. Zaharia M, Chowdhury M, Franklin M J, et al.Spark:Cluster computing with working sets[C/OL]//Proc of the2nd USENIX Conf on Hot Topics in Cloud Computing (HotCloud) .Berkeley, CA:USENIX Association, 2012[2017-01-18].http://static.usenix.org/legacy/events/hotcloud10/tech/full\_papers/Zaharia.pdf
6. Zaharia M, Chowdhury M, Das T, et al.Resilient distributed datasets:A fault-tolerant abstraction for inmemory cluster computing[C]//Proc of the 9th USENIXConf on Networked Systems Design and Implementation.Berkeley, CA:USENIX Association, 2012:15-28
7. Jiang Zhipeng, Chen Haopeng, Zhou Huan, et al.An elastic data persisting solution with high performance for Spark[C]//Proc of the 2015 Int Conf on Smart City/SocialCom/SustainCom.Piscataway, NJ:IEEE, 2015:656-661
8. Rizvi S S, Chung T S.Flash SSD vs HDD:High performance oriented modern embedded and multimedia storage systems[C]//Proc of the 2nd Int Conf on Computer Engineering and Technology (ICCET) .Piscataway, NJ:IEEE, 2010:297-299
9. Narayanan I, Wang Di, Jeon M, et al.SSD failures in datacenters:What, when and why?[C]//Proc of the 9th ACM SIGMETRICS Int Conf.New York:ACM, 2016:407-408
10. Meza J, Wu Qiang, Kumar S, et al.A large-scale study of flash memory failures in the field[C]//Proc of ACMSigmetrics Performance Evaluation Review.New York:ACM, 2015:177-190
11. Ouyang Jian, Lin Shiding, Hou Zhenyu, et al.Active SSDdesign for energy-efficiency improvement of Web-scale data analysis[C]//Proc of the 2013Int Symp on Low Power Electronics and Design.Piscataway, NJ:IEEE, 2013:286-291
12. Schroeder B, Lagisetty R, Merchant A.Flash reliability in production:The expected and the unexpected[C]//Proc of the 14th USENIX Conf on File and Storage Technologies (FAST&#39;16) .Berkeley, CA:USENIX Association, 2016:67-80
13. Gantz J, Reinsel D.Extracting value from chaos, IDCIVIEW[R/OL].Hopkinton, MA:EMC Corporation, 2011[2017-01-18].https://www.emc.com/collateral/analystreports/idc-extracting-value-from-chaos-ar.pdf
14. Canim M, Mihaila G A, Bhattacharjee B, et al.SSDbufferpool extensions for database systems[J].Proceedings of the VLDB Endowment, 2010, 3 (1/2) :1435-1446
15. Kang W H, Lee S W, Moon B.Flash-based extended cache for higher throughput and faster recovery[J].Proceedings of the VLDB Endowment, 2012, 5 (11) :1615-1626
16. Ni Yuanjiang, Jiang Ji, Jiang Dejun, et al.S-RAC:SSDfriendly caching for data center workloads[C]//Proc of the9th ACM Int on Systems and Storage Conf.New York:ACM, 2016:8:1-8:12
17. Awasthi A, Nandini A, Bhattacharya A, et al.Hybrid HBase:Leveraging flash SSDs to improve cost per throughput of HBase[C]//Proc of the 18th Int Conf on Management of Data.Mumbai, India:Computer Society of India, 2012:68-79
18. Luo Tian, Lee R, Mesnier M, et al.hStorage-DB:Heterogeneity-aware data management to exploit the full capability of hybrid storage systems[J].Proceedings of the VLDB Endowment, 2012, 5 (10) :1076-1087
19. SAMSUNG.SAMSUNG GREEN SSD[R/OL].Austin, Texas:SAMSUNG SEMICONDUCTOR, INC, 2010[2017-01-18].http://www.samsung.com/us/business/oem-solutions/pdfs/SSI-green\_ssd\_v4.pdf
20. Li Min, Tan Jian, Wang Yandong, et al.Sparkbench:Acomprehensive benchmarking suite for in memory data analytic platform Spark[C]//Proc of the 12th ACM Int Conf on Computing Frontiers.New York:ACM, 2015:No.53
21. Zhu Jizhao, Jia Yantao, Xu Jun, et al.SparkCRF:Aparallel implementation of CRFs algorithm with Spark[J].Journal of Computer Research and Development, 2016, 53 (8) :1819-1828 (in Chinese) (朱继召, 贾岩涛, 徐君, 等.SparkCRF:一种基于Spark的并行CRFs算法实现[J].计算机研究与发展, 2016, 53 (8) :1819-1828)
22. Bilal M, Oyedele L O, Qadir J, et al.Big data in the construction industry:A review of present status, opportunities, and future trends[J].Advanced Engineering Informatics, 2016, 30 (3) :500-521
23. Doulkeridis C, Nrvg K.A survey of large-scale analytical query processing in MapReduce[J].The VLDB Journal, 2014, 23 (3) :355-380
24. Meng Shunmei, Dou Wanchun, Zhang Xuyun, et al.KASR:A keyword-aware service recommendation method on MapReduce for big data applications[J].IEEE Trans on Parallel and Distributed Systems, 2014, 25 (12) :3221-3231
25. Zhang Xuyun, Yang L T, Liu Chang, et al.A scalable twophase top-down specialization approach for data anonymization using MapReduce on cloud[J].IEEE Trans on Parallel and Distributed Systems, 2014, 25 (2) :363-373
26. Shi Xuanhua, Chen Ming, He Ligang, et al.Mammoth:Gearing Hadoop towards memory-intensive MapReduce applications[J].IEEE Trans on Parallel and Distributed Systems, 2015, 26 (8) :2300-2315
27. Rana N, Deshmukh S.Performance improvement in Apache Spark through shuffling[J].International Journal of Science, Engineering and Technology Research, 2015, 4 (3) :1636-1638
28. Zhao Yao, Hu Fei, Chen Haopeng.An adaptive tuning strategy on Spark based on in-memory computation characteristics[C]//Proc of the 18th IEEE Int Conf on Advanced Communication Technology (ICACT) .Piscataway, NJ:IEEE, 2016:484-488
29. Zhu Wei, Chen Haopeng, Hu Fei.ASC:Improving Spark driver performance with automatic Spark checkpoint[C]//Proc of the 18th IEEE Int Conf on Advanced Communication Technology (ICACT) .Piscataway, NJ:IEEE, 2016:607-611
30. Meng Xiangrui, Bradley J, Yavuz B, et al.Mllib:Machine learning in Apache Spark[J].Journal of Machine Learning Research, 2016, 17 (1) :1235-1241
31. Kulkarni S.A recommendation engine using Apache Spark[D].San Jose, CA:The Faculty of the Department of Computer Science San Jose State University, 2015