DOI:10.3969/j.issn.1009-6868.2016.02.005
网络出版地址:http://www.cnki.net/kcms/detail/34.1228.TN.20160224.1702.004.html

分布式数据处理系统内存对象管理问题分析

In-Memory Data-Object Management in Distributed Data Processing System

张雄/ZHANG Xiong
陆路/LU Lu
石宣化/SHI Xuanhua
(华中科技大学 大数据技术与系统湖北省工程实验室,湖北 武汉 430074)
(Big Data Technology and System Lab, Huazhong University of Science and Technology, Wuhan 430074, China)

中图分类号:TP393 文献标志码:A 文章编号:1009-6868 (2016) 02-0019-004

摘要: 通过从程序语言的特性、垃圾回收机制、内存对象的序列化机制到基于区域的内存管理机制分析了内存对象的管理存在的问题,并分析了内存对象的生命周期在内存对象管理中能发挥的作用。提出了基于内存对象的生命周期对内存进行区域化管理的思路,可以从根本上解决垃圾回收问题。

关键词: 大数据;内存对象管理;分布式数据处理系统

Abstract: Through analysis of the characteristics of program languages, mechanism of garbage collection, serialization of in-memory data-objects and region-based memory management, most existing problems with memory management are exposed. What's more, the lifetime of in-memory data objects can be key factor in memory management. Thus a solution is region-based memory management combined with the lifetime of in-memory data objects, which can solve the garbage collection problem.

Key words: big data; in-memory data-object management; distributed data processing system

以MapReduce为代表的分布式数据处理系统使得人们可以以增加硬件资源的方法来处理海量数据。已有的大量研究集中在多核或分布式环境下的可扩展性和容错性。最新的研究工作表明:这类系统计算执行效率是一个被忽视的重要问题。导致执行效率低下的一个重要原因是代表性的开源系统,如Hadoop和Spark,都使用带有托管执行环境的高级语言开发,从而降低分布式环境下的部署和调试的难度。托管环境提供内建的高级功能,比如自动内存管理和并发模型,使得其对象模型的底层实现非常复杂,带来额外的内存和中央处理器(CPU)开销。工业级托管环境的现代即时编译优化很大程度上解决了中间代码(IR)的解释执行效率问题,但是难以解决复杂对象模型实现带来的对象管理开销问题。以Java虚拟机(JVM)为例:(1)每个对象在JVM中会有一个头结构保存元数据,头结构除了记录对象类型,还要支持垃圾收集和并发加锁优化;(2)所有(非原生类型)对象都在堆中创建,因此每个存活对象至少要有一个额外变量保存其引用;(3)泛型容器的元素如果是基本类型,必须首先被装箱为对象类型;(4)主流垃圾收集算法都是基于对象追踪的,因此堆中有大量的存活对象时,垃圾收集器需要耗费大量CPU周期来标记存活对象[1]。随着处理数据量的增大,内存对象越来越多,尤其是长驻内存对象的存在[2],内存对象管理会带来严重的内存膨胀和CPU开销问题。内存膨胀会间接影响执行性能:如果内存足够,更大的内存占用会导致更频繁的垃圾回收;如果内存不足,缓存的数据需要部分丢弃或者换出到磁盘,导致额外的重计算或输入输出(IO)开销。

最初用来解决这类问题的方法是垃圾收集优化。垃圾收集优化分为两个方面:一方面是通过参数[3]调优,避免频繁垃圾收集;另一方面是通过优化垃圾收集算法实现来提高垃圾收集的性能[4]。垃圾收集优化的方法只是减缓了垃圾收集操作的影响,实际上内存对象管理所存在的问题仍然存在。后续的性能优化方案逐渐从垃圾收集的优化深入到内存对象管理本身,针对数据对象在内存中的存储进行优化,主要包括序列化存储[5],基于区域的内存管理[6]。这种解决方案从根本上解决了对象对内存资源的占用,但是仍然无法避免对

收稿时间:2016-01-23
网络出版时间:2016-02-25
基金项目:国家自然科学基金(61433019、61370104)

象的存在。在对象存储优化的基础上，目前分布式系统的一些上层特定应用，例如 Spark 结构化查询语言（SQL），利用特定的数据结构解决了对象存储优化的不足，从根本上消除对象。当然，由于是特定的应用系统，使用范围窄。

1 垃圾收集优化

垃圾收集问题是内存对象管理中最重要的问题之一，也是影响系统性能的关键因素。因此，最初针对内存对象管理问题的解决方法都是从垃圾收集入手。

1.1 垃圾收集调优

垃圾收集调优是最传统的垃圾收集优化技术，也是一些长时间运行的低延迟 Web 服务所推荐的方法。一些开源分布式数据处理系统，例如 Cassandra 和 HBase 都使用以延迟为中心的方法来避免长时间垃圾收集开销[7]。以上所述垃圾收集调优方法的关键在于：用标记清除算法（CMS）的垃圾收集控制器代替原有以吞吐量为中心的垃圾收集；调整标记清除算法的垃圾收集控制器参数以降低垃圾收集开销。

1.2 垃圾收集算法优化

目前的垃圾收集算法有引用计数法、标记清除算法、拷贝收集算法等，不同的垃圾收集算法让内存对象管理有更多的选择来处理对象的回收。垃圾收集算法的优化性能要到达最优一般有特定场景，例如非统一内存访问（NUMA）感知的垃圾回收器[8]。并且，垃圾收集算法的优化只是掩盖了内存对象管理的问题，内存对象的自动化管理存在的问题仍然存在，频繁调用垃圾收集的本质因素没有解决。

1.3 程序语言优化

分布式数据处理系统使用高级面向对象语言进行开发会导致内存对象管理的问题，而传统的面向机器的语言，如 C、C++，则不存在内存对象管理存在的问题。为了追求性能上的优势，一部分企业机构会选择用这些传统的语言来改写目前的分布式数据处理系统，但是失去了高级语言特性的系统开发难度非常之大，并且不利于系统的更新。

2 对象存储的优化策略

垃圾收集的优化并没有考虑内存对象管理所存在的问题的本质，即内存中的对象仍然是自动化管理的。所以一些系统将对象用序列化的方式存放到内存以减少内存的占用来防止频繁垃圾收集，或者将常规的内存对象管理方法替换为针对对象标记回收的区域内存管理方法来消除垃圾收集。这一类方法从内存对象管理的本质上考虑了性能问题。

2.1 序列化存储

目前的分布式数据处理系统，如 Hadoop 和 Spark，都支持将中间数据对象序列化为 byte 数组，从而减少对象在内存中存储的占用。Hadoop 系统中的对象大部分都是临时的数据对象，因此 Hadoop 仅将 Map 的输出数据序列化成 byte 数据，存放到磁盘，然后通过 Shuffle 传输给 reduce task。尽管不存在内存对象管理的问题，但是序列化机制确实对分布式数据处理系统有重要作用。Spark 系统不仅在 Shuffle 阶段提供序列化机制，还在 Cache 时提供了序列化选择，Cache 时 Spark 会将弹性分布式数据集（RDD）中的数据保存到内存，用户可以选择是否采用序列化保存数据。Spark 之所以支持非序列化保存，是因为序列化机制存在序列化和反序列化的开销。一般来说，序列化机制能够有效降低内存对象的占用，但是要在 Cache 数据对象时执行序列化操作，而在使用对象时执行反序列化操作。

序列化存储降低了内存对象的占用，但是应用仍然基于对象执行的。因此在序列化和反序列化的基础上，内存对象管理仍然需要考虑中间对象的管理，当数据量大时，对象的回收仍然会影响系统的性能[9]。

2.2 基于区域的内存管理

无论是基于垃圾收集调优还是序列化存储，都是由内存管理机制自动标注对象的生命周期，始终存在对象的操作，就必然会需要内存管理机制根据标注回收无需再使用的对象，也就必然会导致垃圾收集。从内存对象相反的一个方向分析，C、C++等语言完全手动的标注内存中使用的对象，手动的回收对象。基于区域的内存管理综合了自动化和完全手动标注内存对象的两种策略，采取了绕过垃圾收集的策略，将一部分对象统一标记后直接存储到堆外区域，整块回收区域内的对象，从而消除频繁垃圾收集，解决内存对象管理的问题。

FACADE[10]系统是基于区域的内存管理的典型实例。FACADE 以程序分析为基础，在程序代码中由用户标识需要转换的 Java 对象。FACADE 会首先识别用户标注的 Java 对象，将其转换为轻量级的 FACADE 对象并通过 byte 形式保存 FACADE 到堆外内存，极大地减少了对象对内存的占用，而 FACADE 相比序列化更加进一步消除对象之处在于它同时转换了 Java 对象的操作代码。用户自定义的操作函数是基于 Java 对象的，FACADE 转换 Java 对象为 FACADE 对象后，同时将操作函数转换为基于 FACADE 对象的操作函数，完全实现了对象的消除。FACADE 的内存对象管理采取了整块分配和整块回收的原则，一方面配合 FACADE 对象的存储方式；一方面减少了垃圾收集开销，相比传统的内存对象管理取得了非常好的效果。

尽管 FACADE 的内存对象管理已经从很大程度上解决了对象管理的垃圾收集问题，但是它基于一个很强烈的假设：在整块分配和整块回收的

操作间隔内的所有对象在内存中的存活时间都是相同的。在一些分布式数据处理系统中，例如 Spark 和 Flink，将作业划分为有向无环图，按照每个阶段执行。这类系统中的数据对象在内存中的存活时间就非常复杂，如果有用户将数据 Cache 到内存，数据对象的存活时间持续整个作业执行期；如果是 Shuffle 阶段的数据对象，数据对象可能会在多个阶段的执行期内都存活在内存中。所以，FACADE 在内存对象的管理上忽略了内存对象的生命周期。

Broom[11]综合考虑了基于区域的内存管理的特点以及内存对象的生命周期的特点，Broom 以 NET CLR 平台为研究对象，将内存对象的存放区域进一步区分为：可转移区域，用来存放操作的传递消息，同一时间只有一个操作可以访问该区域；操作所需区域，针对某个操作私有的对象存储区域，该区域内的对象的生命周期与相应的操作生命周期相同；临时区域，存放一些临时的数据对象。由于 Broom 基于闭源的系统实现而且作为 short paper 对系统实现提及较少，所以能够获取的信息只在于基于区域的内存管理与内存对象生命周期特点的结合是消除垃圾收集所必须考虑的两个因素。

3 特定应用领域的优化

大多优秀的开源分布式数据处理系统，如 Hadoop 和 Spark，都基于底层的系统实现了上层应用领域的生态系统[12]，例如 Spark 生态系统包括 Spark SQL、Spark Streaming、Spark GraphX 和 Spark MLlib。特定应用领域的分布式数据处理系统在底层数据系统的基础上定义了特定的计算结构，从而可以实现更加复杂的内存对象管理机制。在最初 Java 等高级语言被应用在数据分析系统时，除了受益于高级语言的特性，一些系统也意识到其在内存对象管理上的不足，因此结合特定的结构进行优化[13]，例如 SQL。Shark 等针对 data-intensive 的数据库管理系统，将 Java 对象转化为 Telegraph 数据流，在内存分配上整存整取，绕过 Java 的内存管理方法[14]。Shark 使用基于列的内存存储和动态查询优化来提高 SQL 查询的性能。Apache 项目 Tungsten 基于 Spark SQL 实现，将传统的关系表结果转换为以列为结构的字节序列，同时将 SQL 操作全部转换为基于字节序列的操作。

分布式数据处理系统内存对象管理的解决方案各具优点和缺点，具体见表 1。

4 结束语

分布式数据处理系统按照数据流的路径可以分为控制路径和数据路径，控制路径由系统框架支持和实现，数据路径由用户定义和实现。我们发现：控制路径的编程实现更多的从对象模型的高级特性中获益，包括类型的运行时动态识别、并发同步操作的偏向锁优化和自动对象内存管理等。数据路径的编程实现很少使用语言的高级特性。数据路径的实现由用户自定义，具体包括用户自定义类型（UDT）和用户自定义方法（UDF）。UDT 定义的是数据路径中实际操作的对象类型，而 UDF 定义了对这些数据类型执行的操作。UDT 通常是基本类型的浅层组合和常用方法的实现封装，很少会使用复杂的继承层次和多态。

比如，通过定义接口来抽象化模块之间的交互，进而通过工厂模式，依赖注入来支持灵活的模块和插件加载，而这些设计模式依赖于多态和反射等语言特性。框架负责任务的并行化执行和同步机制，因此依赖于托管环境的并发执行模型，比如线程创建和加锁操作。由于并发执行由框架负责，UDF 本身都是串行代码，在 UDF 内部使用加锁操作通常没有意义，更不可能会在 UDT 数据对象上加锁。因此，UDT 并不依赖于一个复杂的对象模型实现。比如：（1）当没有多态导致的虚方法派发时，也没有使用反射时，不需要在对象头部记录对象的运行时类型；（2）当对象没有加锁操作时，也不需要在头部存储偏向锁状态；（3）UDT 数据对象的生命周期具有很强的规律性，如果能够跳过 JVM 的内存管理，不仅消除了垃圾收集的 CPU 开销，也不需要在对象头部存储垃圾收集所需的状态信息。

最重要的一点是：数据对象的生命周期具有很强的规律性。我们发现：在以 Spark 为代表的新一代通用数据并行系统中，作业执行时通常有几类数据容器会持有数据对象，而数据对象的生命周期和持有它的数据容器的生命周期有很强的关联性：

（1）UDF 变量。包括 UDF 对象的字段和 UDF 局部变量，前者的生命周

▼表 1 3种内存对象管理问题的解决方案对比分析

解决方案		特点	优点	缺点
垃圾收集优化	垃圾收集调优	调整系统运行参数，减少垃圾收集频率	适用范围广，目前系统均适用	调节能力有限，不可避免垃圾回收
	垃圾收集算法优化	优化垃圾收集算法，加快垃圾收集过程	一种优化算法可以适用多种系统	没有从系统层面解决内存对象的管理问题
	程序语言优化	利用语言特性手动释放占用的内存	从语言层面根本解决了内存中对象的管理问题	面向对象语言的特性不复存在，系统开发难度大
对象存储优化	序列化存储	利用序列化器减少内存中对象的占用	精简了内存中的对象形式，一定程度上缓解了垃圾收集的影响	数据量大而内存有限时，仍然无法解决内存对象的管理问题
	基于区域的内存管理	堆外区域整块标记、分配和回收	综合了自动回收内存空间和面向对象的自动内存管理的优点，减少了频繁了垃圾回收	需要用户手动标记，影响用户程序编写；区域的生命周期复杂时适用程度有限
特定领域优化		利用上层特定系统的特定数据结构优化内存中的对象	特定的数据结构可以极大程度上精简内存中的对象存储，也非常适合进一步在流程上优化	优化策略的通用性有限，只适用于一类特定的系统

期刚好为一个任务的执行时间，其存活时间内时所持有的对象生命周期由该字段的赋值操作决定，但最长不超过任务的执行时间；后者的生命周期为一次UDF方法的调用，因此其持有的数据对象最多存活一次方法调用，可以视为临时对象。

（2）缓存数据集。缓存数据集的生命周期由应用程序显示决定，其持有的数据对象的生命周期具有与该RDD等同的生命周期。

（3）Shuffle缓冲区。不考虑溢出到磁盘的情况，Shuffle缓冲区的生命周期也刚好为一个任务的执行时间。其持有的数据对象的生命周期较为复杂，在聚合计算的过程中，Shuffle buffer仅为一个Key数据对象保存一个数据对象作为当前聚合的结果。

一种可行的方法是通过自动转换数据处理应用程序来减少程序运行时创建的UDT数据对象的数量。转换工作对应用开发人员完全透明，不会限制数据并行编程模型的表达能力和灵活性。

致谢

本研究得到华中科技大学金海教授的指导，谨致谢意！

参考文献

[1] JONES R, HOSKING A, MOSS E. The Garbage Collection Handbook : the Art of Automatic Memory Management [M]. USA: CRC Press, 2012
[2] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient Distributed Datasets: a Fault–Tolerant Abstraction for in–Memory Cluster Computing [C]//Proceeding in 9th USENIX Conference on Networked Systems Design and Implementation (NSDI). USA: USENIX Association, 2012：141–146
[3] Cassandra Garbage Collection Tuning, Find and Fix Long GC Pauses [EB/OL]. [2013–11–14]. http://aryanet.com/blog/cassandra–garbage–collector–tuning
[4] Laboratory for Web Algorithmic [EB/OL]. [2014–10–12].http://law.di.unimi.it/datasets.php
[5] CARPENTER B, FOX G, KO S H, et al. Object Serialization for Marshalling Data in a Java Interface to MPI[C]//ACM Java Grande Conference. USA: ASM, 1970：66–67
[6] MADS T, JEAN–PIERRE T. Region–Based Memory Management [J]. Information & Computation, 1997, 132(2):109–176
[7] The Garbage Collector and Apache Hbase [EB/OL]. (2016–02–18)[2014–03–22].http://hbase.apache.org/book.html#gc
[8] 浦云明. JAVA垃圾收集器算法分析及垃圾收集器的运行透视[J]. 计算机系统应用, 2003 (11): 39–41
[9] MILLER H, HALLER P, BURMAKO E, et al. Instant Pickles: Generating Object–Oriented Pickler Combinators for Fast and Extensible Serialization [J]. ACM Sigplan Notices, 2013, 48(10):183–202
[10] NGUYEN K, WANG K, BU Y, et al. FACADE: A Compiler and Runtime for (Almost) Object–Bounded Big Data Applications [J]. ACM Sigplan Notices, 2015, 50(4):675–690. DOI: 10.1145/2775054.2694345
[11] GOH I, GICEVA J, SCHWARZKOPF M, et al. Broom: Sweeping Out Garbage Collection from Big Data Systems [C]//15th Workshop on Hot Topics in Operating Systems (HotOS XV). USA: ACM, 2015
[12] 胡俊, 胡贤德, 程家兴. 基于Spark的大数据混合计算模型[J]. 计算机系统应用, 2015(4): 214–218
[13] SHAH M A, FRANKLIN M J, MADDEN S, et al. Java Support for Data–Intensive Systems: Experiences Building the Telegraph Dataflow System [J]. Sigmod Record, 2001, 30(4):103–114
[14] XIN R. S, ROSEN J, ZAHARIA M, et al. Shark: SQL and Rich Analytics at Scale[C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. USA: ASM, 2013：13–24. DOI: 10.1145/2463676.2465288

作者简介

张雄，华中科技大学在读硕士；主要研究领域为分布式数据处理系统。

陆路，华中科技大学在读博士；主要研究领域为分布式数据处理系统。

石宣化，华中科技大学教授；主要研究领域为并行计算与分布式系统。

综合信息

光纤到户堡垒拆除：网速直通10G

最近，英国伦敦大学设计并测试了一种新型光接收机，有望大大降低光纤网络直达家庭用户的成本，使每个家庭直接与全球互联网相连。

据物理学家组织网14日报道，FTTH通常只到交接箱，还远不及终端用户。所谓的“最后1公里”，即家庭用户通过交接箱与全球互联网的连接，大多是用铜缆，但能读取光信号的光接收机非常昂贵，许多家庭难以负担。即使在FTTH技术领先的日本、韩国等国家，FTTH连接也不足50%，在英国还不到1%。

限制FTTH的主要原因是成本，要实现它不仅要把光缆铺到每个家庭，还要提供用户负担得起的光接收机。英国伦敦大学光网团队和其他团队共同合作开发的新型光接收机，保留了传统光接收器的诸多优点，但体积更小，只有原来75%~80%的组件，显著降低了制造成本和维修成本，而且它的灵敏度能与现有的网络相匹配。

（转载自《中国信息产业网》）