

大数据场景下语言虚拟机的性能分析与优化调研

张正君, 余博识, 贾兴国

Shanghai Jiao Tong University

{zhangzhengjunsjtu, 201608ybs, jiaxg1998}@sjtu.edu.cn

School of Software Engineering

版本: 1.0

日期: 2020 年 12 月 15 日

摘 要

随着大数据处理系统的快速发展, 托管型语言 (managed languages) 由于其快速的开发周期以及丰富的社区资源得到了广泛的应用。例如, 分布式数据处理框架 Spark、Hadoop、DryadLINQ、分布式文件系统 HDFS、分布式键值存储系统 HBase、Cassandra 等均使用开发效率较高的 Java 语言进行开发。托管型语言通过使用语言虚拟机实现对内存的管理, 如使用 JVM (Java Virtual Machine) 进行垃圾回收 (garbage collection, GC), 免去手动编写代码进行内存释放的需要, 也降低了内存管理出现错误的可能。然而, 开发的便利性也带来了一定的性能损耗。经过调研, 研究者们从不同的角度发现了托管型语言在分布式大数据处理场景下出现的性能问题, 如 GC 时间过长、节点间数据传输太慢、语言虚拟机启动耗时过长等, 并且提出了相应的解决方案。本文总结了研究者们发现的问题及其解决方案, 并且针对解决方案的局限性与适用性进行了分析, 提出了相应的优化方案。

关键词: 语言虚拟机, 分布式系统, 大数据, 垃圾回收

目录

1 研究背景	3
2 大数据场景下语言虚拟机的性能问题与优化	3
2.1 分布式环境下 GC 对性能的影响	4
2.1.1 问题概述	4
2.1.2 解决方案	4
2.2 JVM 启动耗时对延迟敏感型应用的影响	4
2.2.1 问题概述	4
2.2.2 解决方案	5
2.3 分布式环境下多个 JVM 的协调问题	5
2.3.1 问题概述	5
2.3.2 解决方案	5
2.4 分布式环境下节点数据交换的性能问题	5
2.4.1 问题概述	5
2.4.2 解决方案	6
2.5 资源分散架构下的 GC 算法优化	6
2.5.1 问题概述	6
2.5.2 解决方案	6
3 HotTub 研究思路分析	6
3.1 JVM 预热开销测试与分析	6
3.2 HotTub 设计与实现	6
3.3 HotTub 性能测试	6
3.4 HotTub 适用性与局限性分析	6
4 HotTub 局限性优化	7
5 总结	7
附录	8
A 附录	9
A.1 //	9

1 研究背景

随着摩尔定律的终结，单个机器所提供的计算资源已经无法满足海量数据的巨大算力需求。学术界和工业界将关注点从纵向扩展转向了横向扩展，使用多台配置较低的廉价机器代替了配置较高的单台机器，使用分布式系统向数据处理、机器学习、云计算等应用提供可横向扩展的计算资源。运行在分布式系统上的数据处理框架如 Spark^[1]、Hadoop^[2]、DryadLINQ^[3]、Naiad^[4]等采用了分布式并行的数据处理模式，充分利用分布式集群提供的计算资源。也有分布式文件系统 HDFS^[5]、分布式键值存储系统 HBase^[6]、Cassandra^[7]等利用分布式系统较强的容错性、可扩展性，为数据存储提供了更优的解决方案。

运行在云平台上的分布式软件的快速发展使得托管型编程语言成为了开发人员和研究人员的关注重点。广泛使用的分布式处理框架 Hadoop、Spark、Cassandra 均使用 Java 进行开发，DryadLINQ、Naiad 等数据处理框架使用由微软开发的 C# 语言进行编写。还有更多的托管型语言被广泛使用，如 Go、Javascript、Python 等。托管型语言如 Java、C# 使用语言虚拟机进行自动内存回收，免去了手动编写代码释放内存的需要，还有面向对象的编程风格、丰富的社区资源，使得 Java 等语言有很高的开发效率。

语言虚拟机带来的开发便捷性建立在一定的性能损耗之上。例如为了运行 Java 程序，首先对 Java 源代码进行编译，生成 Bytecode（字节码），再从 classpath 中加载、解析、验证字节码文件，以及运行该字节码所需的其他字节码文件，后在 JVM 中运行字节码。Java 语言运行环境（runtime）的加载以及字节码的解释执行相比于 Native 的编程语言（如 C、C++）产生了巨大的开销。同时，Java 程序在堆（heap）内存中为对象申请内存，无需手动编写代码释放堆内存，而是使用 JVM 自带的 GC（垃圾回收）功能进行对内存的自动释放。然而，进行 GC 的过程中，语言虚拟机需要占用应用的 CPU 资源，或需要应用暂停执行（Stop-the-world），这在延迟敏感的情境下对应用的性能表现有巨大的影响。在分布式数据处理的场景中，由于 Java 等管理型语言使用在堆上保存的对象（Object）来存储、处理数据，GC 需要遍历海量的对象，其开销相对于单机、数据量较小的情景也被明显加大。分布式环境下请求的处理依赖于多个节点上的语言运行环境，语言虚拟机的性能优化需要依靠多个语言虚拟机之间的协调，这和单机情境下的语言虚拟机性能优化又大有不同。

本文针对大数据处理场景下的语言虚拟机性能优化进行调研，总结出了分布式大数据处理对语言虚拟机提出的新的挑战，以及研究者所提出的解决方案，并分析了特定解决方案的适用性与优化方案。本文第二章总结了语言虚拟机在大数据情境下表现出的性能问题，以及研究者对相关问题所提出的解决方案。第三章就 JVM 启动耗时过长问题的研究思路进行了分析，并总结了其局限性与适用范围。第四章提出了针对第三章中适用性问题的解决方案，并搭建原型、验证和测试了解决方案的有效性。在第五章中我们对本文的调研工作进行总结。

2 大数据场景下语言虚拟机的性能问题与优化

大数据处理为托管型语言带来了全新的使用场景，同时托管型语言为大数据处理框架带来了较高的开发效率。新的使用情景为托管型语言的语言虚拟机带来了新的挑战。本章对大数据环境下语言虚拟机表现出的性能问题做了总结分析，并对研究者的相应解决方案进行了总结。

2.1 分布式环境下 GC 对性能的影响

2.1.1 问题概述

在大数据场景下，应用需要良好的性能与扩展性，然而托管型语言的运行时环境存在臃肿、可扩展性差的问题。语言虚拟机的 GC 使得应用的主线程必须停止，需要遍历堆中保存的相互引用的复杂的对象，占用大量计算资源，无法进行有用的工作；内存管理模块也有巨大的额外内存开销，虽然所处理的数据量大小比堆内存的容量更小，也会产生 OOM (Out Of Memory, 内存不足) 的报错。但使用非管理型语言会增加应用编写者的负担，更容易出现错误，调试内存管理错误也是一件非常困难的事。当前大量的大数据处理框架已经使用管理型语言进行编写，向非管理型语言的迁移会带来巨大的工作量。故需要有一种方法优化语言虚拟机的 GC 问题与内存空间占用问题。

2.1.2 解决方案

Nguyen 等人提出了 Facade^[8] 编译框架。他们认为将大量数据保存在堆内存上，让 JVM 进行管理是不明智的。通过限制堆内存上保存的对象数量，将数据面与控制面进行分离，仅在堆上保存控制面对象，而数据面中的数据保存在本地内存 (native memory) 中进行手动管理。这打破了面向对象编程语言的一条规则：对象应当包含数据以及操作数据的接口。对于每一个 Java 语言中的堆上保存的对象，Facade 编译系统生成一个相应的 facade，提供对象的操作接口而不保存对象中包含的数据，从而减轻垃圾收集器的负担，而程序员仅需标注出数据类 (data class)。对于 facade，可以通过对象复用限制堆中对象的数量；而 native memory 中的数据在每个 iteration 结束后统一进行释放，无需访问数据的每个字段，相比于垃圾收集器有更优的性能。经测试，Facade 使得应用运行时间缩短了 48%，降低了 50% 的内存占用。

Gog 等人提出了 Broom^[9] 内存管理系统，取代了 CLR 的 GC 系统。他们认为，大数据系统中存在多个 Actor，这些 Actor 独立地运行，它们之间传递消息，形成了高度结构化的数据流，例如 MapReduce^[10] 中的 map 工作线程与 reduce 工作线程。每个 Actor 内的对象在 Actor 完成工作后不再会用到，仅有 Actor 之间传递的数据需要保存，无需 GC 扫描。为了针对这类高度结构化的应用对垃圾收集器进行优化，Broom 内存管理器使用了基于 region (分区) 的内存管理策略，对象在各个 region 中保存而不在堆中由 GC 管理，由程序员手动释放 region 中的对象。Broom 内存管理器减轻了 GC 的负担，减少了 59% 的运行时间，但使得程序员的负担增加。

2.2 JVM 启动耗时对延迟敏感型应用的影响

2.2.1 问题概述

社区中长期存在对 Java 性能的激烈讨论，尤其是 Java 是否适合编写延迟敏感型应用的问题。一些程序员坚持使用 C++ 等非管理型语言编写键值存储系统，因为他们认为 Java 天生就是较慢的。Java 之所以适用于编写 Hadoop 框架是因为 Hadoop 大部分时间在进行 I/O 操作。在学术界，对于大数据框架的性能优化主要集中在对 GC 性能的优化、调度策略的优化、数据交换开销的优化等，故我们缺少对 Java 各个方面性能的整体了解。经过实验测得，在 I/O 密集型任务

中，JVM 的预热开销也是较大的，如在从 HDFS 中读取 1G 的文件的过程中，JVM 预热使用的总时间占运行时间的 33%，预热时间不随任务数据量的变化而变化。在大数据框架中，复杂的软件栈使得 Java 运行环境的加载更加缓慢，因为需要加载更多的 class，如 Spark 为了完成一次请求需要加载 19066 个 class。

2.2.2 解决方案

Lion 等人经过实验测得，JVM 的性能瓶颈在于 JVM 的启动预热（warm-up），并编写了 HotTub 代替了 HotSpot^[11] 解决了 Java 虚拟机启动时间过长的的问题。他们首先通过修改 Java 程序的调用过程测量出 class 加载与字节码解释执行的开销，观察出 JVM 预热对性能的巨大影响。同时，他们在 HotSpot 的基础上修改实现了 HotTub，在多次请求中复用同一个 JVM 实例，从而免去 JVM 的预热开销。由于大数据应用的相似性，JVM 有较大的复用可能，同时频繁使用的字节码在多次复用后被 JIT 编译成机器码而无需解释执行。HotTub 可以让 Java 程序无修改地运行，免去了 JVM 的预热开销，使得 Spark 请求的性能提升到 1.8 倍。本文将在第三章对 Lion 等人的 HotTub 研究进行详细的分析，总结其使用范围与局限性，并在第四章中提出并测试相应的优化方案。

2.3 分布式环境下多个 JVM 的协调问题

2.3.1 问题概述

在分布式大数据处理框架中，工作负载运行在多个节点上，即运行在多个相互独立的运行时系统（语言虚拟机）上。这会成为性能降低的原因之一，因为每个运行时系统仅拥有当前运行节点的运行状况，而非全局的分布式应用的运行状况。这是之前的工作无法解决的问题，之前的工作仅针对单个节点上的运行时系统进行性能优化。例如，某个节点在不合适的时间进行 GC，导致该节点成为整个请求的拖后腿者，虽然其余节点在较短时间内完成了自己的工作，但仍需等待最慢的节点完成 GC，才能将请求结果返回给用户。这将会对延时敏感型与面向吞吐量的应用都造成性能影响。

2.3.2 解决方案

Maas 等人设计的 Taurus^[12] 系统通过在分布式处理系统中加入一个分布式决策机制，将每个节点上独立的运行时系统组织为一个 Holistic Runtime System（全局运行时系统），从而做出在分布式系统全局下正确的 GC、JIT 决策。每个节点上运行的 Runtime system 定期向 Raft 一致性系统中发送当前的运行状态，// TODO 完善介绍

2.4 分布式环境下节点数据交换的性能问题

2.4.1 问题概述

在大数据系统中，节点间数据的传递是频繁发生的。和之前 GC 性能问题的根源相同，数据传递也因为需要处理海量的包含着数据的对象（Object）而受到了性能影响。数据发送方需要对

于 Java 中的对象 (Object) 进行 *serialize* (序列化) 才可以在网络中传递, 接收方需要将接收到的字节流进行 *deserialize* (反序列化) 才能生成发送方想要发送的对象。序列化和反序列化严重依赖于 Java 中的反射机制, 这是一个开销较大运行时操作, 会严重影响应用的性能; 同时程序员需要手写序列化和反序列化函数, 较容易出现错误。这要求我们进一步考虑 Java 等面向对象语言中 Object 与数据处理的关系。// TODO 完善

2.4.2 解决方案

Nguyen 等人提出的 Skyway^[13] 将本地与远程的 JVM 进程中的堆进行连接, 使得源节点的堆中的对象无需序列化即可传输到目的节点 // TODO 完善

Navasca 等人认为, 数据分析的任务经常使用的数据类型是不可变且被限制的, 他们开发的 Gerenuk^[14] 编译器 // TODO 完善

2.5 资源分散架构下的 GC 算法优化

2.5.1 问题概述

在大数据处理集群中, 资源碎片化问题影响着集群资源使用率, 如内存资源, 单个节点不一定在全部的时间段都对内存有较高的利用率, 当内存利用率较低时, // TODO 待完善

2.5.2 解决方案

Chenxi Wang 等人开发的 Semeru^[15] 是一个分布式的 JVM // TODO

3 HotTub 研究思路分析

3.1 JVM 预热开销测试与分析

// TODO

3.2 HotTub 设计与实现

// TODO

3.3 HotTub 性能测试

// TODO

3.4 HotTub 适用性与局限性分析

// TODO

4 HotTub 局限性优化

// TODO

5 总结

// TODO

参考文献

- [1] APACHE. Apache spark - unified analytics engine for big data[M]. Website: <http://spark.apache.org>, 2020.
- [2] APACHE. Apache hadoop[M]. Website: <http://hadoop.apache.org>, 2020.
- [3] YU Y, ISARD M, FETTERLY D, et al. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language[C/OL]//DRAVES R, VAN RENESSE R. 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings. USENIX Association, 2008: 1-14. http://www.usenix.org/events/osdi08/tech/full_papers/yu_y/yu_y.pdf.
- [4] MURRAY D G, MCSHERRY F, ISAACS R, et al. Naiad: a timely dataflow system[C/OL]//KAMINSKY M, DAHLIN M. ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013. ACM, 2013: 439-455. <https://doi.org/10.1145/2517349.2522738>.
- [5] APACHE. Hadoop distributed file system (hdfs)[M]. Website: http://hadoop.apache.org/docs/stable/hdfs_design.html, 2020.
- [6] APACHE. Apache hbase[M]. Website: <http://hbase.apache.org>, 2020.
- [7] APACHE. Cassandra[M]. Website: <http://cassandra.apache.org>, 2020.
- [8] NGUYEN K, WANG K, BU Y, et al. FACADE: A compiler and runtime for (almost) object-bounded big data applications [C/OL]//ÖZTURK Ö, EBCIOGLU K, DWARKADAS S. Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015. ACM, 2015: 675-690. <https://doi.org/10.1145/2694344.2694345>.
- [9] GOG I, GICEVA J, SCHWARZKOPF M, et al. Broom: Sweeping out garbage collection from big data systems[C/OL]//CANDEA G. 15th Workshop on Hot Topics in Operating Systems, HotOS XV, Kartause Ittingen, Switzerland, May 18-20, 2015. USENIX Association, 2015. <https://www.usenix.org/conference/hotos15/workshop-program/presentation/gog>.
- [10] DEAN J, GHEMAWAT S. Mapreduce: simplified data processing on large clusters[J/OL]. Commun. ACM, 2008, 51(1): 107-113. <http://doi.acm.org/10.1145/1327452.1327492>.
- [11] LION D, CHIU A, SUN H, et al. Don't get caught in the cold, warm-up your JVM: understand and eliminate JVM warm-up overhead in data-parallel systems[C/OL]//KEETON K, ROSCOE T. 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016. USENIX Association, 2016: 383-400. <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/lion>.
- [12] MAAS M, ASANOVIC K, HARRIS T, et al. Taurus: A holistic language runtime system for coordinating distributed managed-language applications[C/OL]//CONTE T, ZHOU Y. Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Atlanta, GA, USA, April 2-6, 2016. ACM, 2016: 457-471. <https://doi.org/10.1145/2872362.2872386>.
- [13] NGUYEN K, FANG L, NAVASCA C, et al. Skyway: Connecting managed heaps in distributed big data systems[C/OL]//SHEN X, TUCK J, BIANCHINI R, et al. Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018. ACM, 2018: 56-69. <https://doi.org/10.1145/3173162.3173200>.
- [14] NAVASCA C, CAI C, NGUYEN K, et al. Gerenuk: thin computation over big native data using speculative program transformation[C/OL]//BRECHT T, WILLIAMSON C. Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019. ACM, 2019: 538-553. <https://doi.org/10.1145/3341301.3359643>.
- [15] WANG C, MA H, LIU S, et al. Semeru: A memory-disaggregated managed runtime[C/OL]//14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 2020: 261-280. <https://www.usenix.org/conference/osdi20/presentation/wang>.

A 附录

A.1 //