Introduction to Garbage Collection Tuning

Wednesday, January 17, 2018 5:41 PM

如今，各种各样的应用程序，小到桌面上的applet，大到服务器上的web service都在使用Java SE。而为了支持这些形形色色的部署，Java HotSpot VM提供了数种垃圾收集器，并且每一种都被设计成可以满足不同的需求。Java SE会根据应用程序运行的计算机类型选择最合适的垃圾收集器。但是，对于每个应用程序，这种选择可能并不是最优的。对于那些有严格的性能目标或其他需求的用户、开发人员和管理员可能需要显式地选择垃圾收集器并调优某些参数以达到预期的性能水平。该文档可以帮助你完成这些任务。

首先，垃圾收集器的一般特性和基本的调优选项都是基于串行的、STW（Stop-The-Word）收集器的上下文中描述的。然后，其他收集器的特定特性，以及考虑因素将会在选择收集器时一起介绍。

什么是垃圾收集器？

垃圾收集器(GC)会自动管理应用程序的动态内存分配请求。

垃圾收集器通过以下操作实现自动动态内存管理:

- 在操作系统中分配和释放内存。
- 分配内存给请求的应用程序。
- 确定哪些内存依然处于被应用程序使用的状态。
- 回收并再分配未被使用的内存。

Java HotSpot垃圾收集器采用了各种技术来提高这些操作的效率:

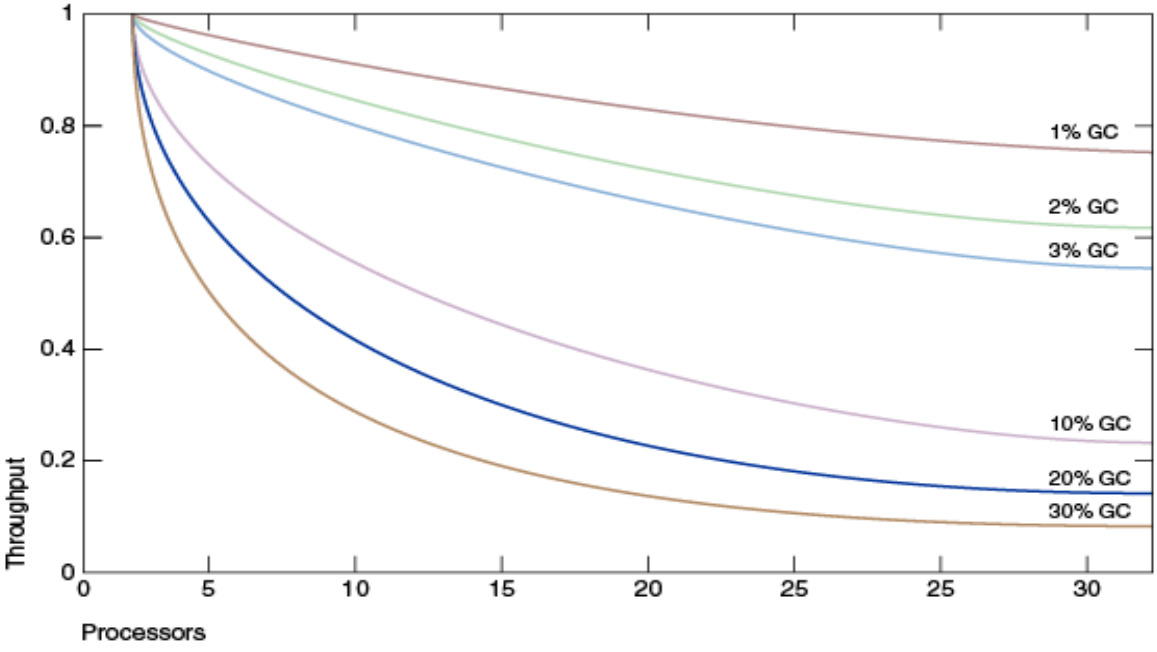
- 将世代清除与老化结合使用，将精力集中于堆内存中最有可能包含大量可回收内存的区域。
- 使用多个线程积极地将操作并行化，或者在后台并发地执行一些长期运行的操作。
- 通过压缩活动对象来尝试恢复更大的连续空闲内存。

为什么垃圾收集器的选择很重要?

垃圾收集器的目的是将应用程序开发人员从手动的动态内存管理中解放出来。开发人员不再需要匹配内存的分配与重分配，并密切关注已分配的动态内存的生命周期。尽管有额外的运行时开销作为代价，但这完全消除了一些与内存管理相关的错误。Java HotSpot VM提供了一系列垃圾收集算法的选择。

选择垃圾收集器什么时候很重要?对于某些应用程序，答案是永远不会。也就是说，应用程序可以在垃圾收集的情况下，在适当的频率和持续时间暂停的情况下表现良好。然而，对于大量的应用程序，特别是那些有大量数据(千兆字节以上)、多线程和高事务率的应用程序来说，情况并非如此。

根据Amdahl法则(在给定的问题中，并行加速会被问题的顺序所限制)意味着大多数工作负载并不能完全并行化;有些部分总是顺序的，并不能从并行中获益。在Java平台中，目前有4个支持的垃圾收集器备选方案，除了Serial GC，其他的都可以并行化工作以提高性能。尽可能降低垃圾收集的开销是非常重要的。图1-1中的图形是一个理想的系统，除了垃圾收集它完全可伸缩。红线是一个应用程序在单处理器系统的垃圾收集中只花费1%的时间。但在有32个处理器的系统中，这导致吞吐量的损失超过20%。品红色的线显示，应用程序在垃圾收集中花费10%的时间(在单处理器应用中，这并不被认为是相当多的时间)，但系统扩展到32个处理器时，会损失超过75%的吞吐量。



这一数据表明，吞吐量问题在小型系统上开发时可以忽略，但扩展到大型系统

时却可能成为主要的性能瓶颈。而针对这个瓶颈的小改进可能带来性能上的飞跃。因此，对于一个足够大的系统，选择合适的垃圾收集器并在必要时对其进行调优是物有所值的。

串行收集器通常适合于大多数小型应用程序，特别是那些在现代处理器上只占用大约100MB的应用程序。其他收集器因为某些专门化的行为，会带来额外的开销或复杂性。如果应用程序不会用到这些专门化行为，使用串行收集器就可以了。在具有大量内存和两个或多个处理器的机器上运行的大型、密集线程应用程序时，串行收集器就不是一个很好的选择。当应用程序运行在这样的服务器类机器上时，默认情况下选择Garbage-First (G1)收集器，看下一章（Ergonomics）。

文档支持的操作系统

该文档及其建议适用于所有JDK 9支持的系统配置，这些配置受到某些垃圾收集器在特定配置中的实际可用性的限制。参见
Oracle JDK 9 and JRE 9 Certified System Configurations.