Ergonomics

Wednesday, February 7, 2018 2:05 PM

Ergonomics是Java虚拟机(Java Virtual Machine, JVM) 和垃圾收集启发式(例如基于行为的启发式方法)改进应用程序性能的过程。

JVM为垃圾收集器、堆大小和运行时编译器提供与平台相关的默认选择。这些选择符合不同类型的应用程序的需求，同时需要更少的命令行调优。此外，基于行为的优化动态优化堆的大小，以满足应用程序的指定行为。

本节描述这些默认选项和基于行为的调优。在使用后面描述的更详细的选项之前，可以使用这些默认值。

垃圾收集器，堆与运行时编译的默认选择

下面是关于垃圾收集器、堆大小和运行时编译器默认选项的信息。

下面这种机器一般被定义为服务器类机器（以下简称服务器）

- 2个及以上物理处理器。
- 2G及以上物理内存。

在服务器上（-server），下面是JVM的默认选项:

- garbage first(G1) 垃圾收集器。
- 初始堆大小为物理内存的1/64。
- 堆大小最大为物理内存的1/4。
- 分层编译器，使用C1和C2。

基于行为的调优

可以将Java HotSpot VM垃圾收集器配置为优先满足两个目标之一:
最大化暂停时间和程序吞吐量。如果满足了首选目标，收集器将尝试最大可能满足另一个目标。当然，这些目标并不总是能够满足的:
应用程序要求一个至少能保存所有的实时数据的最小的堆，而其他配置可能会妨碍实现某些或全部的目标。

最大化暂停时间

暂停时间是垃圾收集器暂停应用程序并回收不再被使用的空间的时间。
最大暂停时间的目的是限制停顿的最大时长。

暂停时间的均值和均值的方差由垃圾收集器维护。均值是在执行开始时取的，但它是加权的，越近的停顿所占的比重会越大。如果暂停时间的平均值加上方差大于其最大目标，那么垃圾收集器就认为没有实现目标。

可以用命令行选项-XX:MaxGCPauseMillis=<*nnn*>来指定最大暂停时间目标。这会被解释为对垃圾收集器的提示，即希望暂停时间为<*nnn*>乃至更少。垃圾收集器会调整堆大小和其他与垃圾收集相关参数，以使垃圾收集暂停时间比目标<*nnn*>短。最大暂停时间目标的默认值因收集器而异。这些不同可能导致更频繁的垃圾收集，从而降低应用程序的总体吞吐量。因此，在某些情况下，期望的暂停时间目标无法实现。

吞吐量

吞吐量目标是以垃圾收集的时间来度量的，而在垃圾收集之外的时间则被称为应用程序时间。
可以通过命令行选项-XX:GCTimeRatio=*nnn*指定吞吐量目标。这样的话，垃圾收集时间与应用时间的比值就是1/ (1+*nnn*)。比如，-XX:GCTimeRatio=19设置了垃圾收集时间目标为总时间的1/20或5%。
垃圾收集时间是所有垃圾收集引起停顿的总时间。如果吞吐量目标没有实现，那么垃圾收集器很有可能增加堆的大小，这样在收集暂停期间在应用程序中花费更长时间。

内存使用

如果吞吐量和最大暂停时间目标都被满足，那么垃圾收集器将减少堆的大小，直到一个目标(始终是吞吐量目标)无法实现。垃圾收集器可以使用的最小和最大堆大小可以使用-Xms=<*nnn*>和-Xmx=<*mmm*>来分别指定。

调优策略

堆增长或缩小到一个足够满足所选择的吞吐量目标的大小。了解堆调优策略，例如选择堆大小的最大值，并选择最大暂停时间目标。

除非你知道你需要一个大于默认最大堆大小的堆，否者不要为堆去选择一个最大值。选择一个对你的应用程序来说恰如其分的吞吐量目标。

应用程序行为的改变会导致堆增长或收缩。比如，如果应用程序开始以更高的速率分配内存，那么堆就会增长以保持吞吐量不变。

如果堆已经增长到最大大小，但吞吐量目标没有满足，那么最大堆大小对于吞吐量目标来说就太小了。将最大堆大小设置为与平台上的物理内存接近的值，但不会导致应用程序的进程切换。再次执行应用程序。如果吞吐量目标仍然没有实现，那么应用程序的时间目标对于平台上可用的内存来说太不切实际了。

如果吞吐量目标可以实现，但是暂停时间太长，那么选择一个最大暂停时间的目标。选择一个最大的暂停时间目标可能意味着你的吞吐量目标无法实现，所以选择对应用程序来说可以接受的折衷值。

当垃圾收集器试图满足相互竞争的目标时，堆的大小会振荡。即使应用程序达到了稳定状态，这种情况依然存在。实现吞吐量目标(可能需要更大的堆)的压力与最大暂停时间和最小内存占用(可能需要小堆)的目标互相竞争。