淘宝主站Cgroup资源控制实践

朱延海（高阳）
核心系统研发部
2012-07

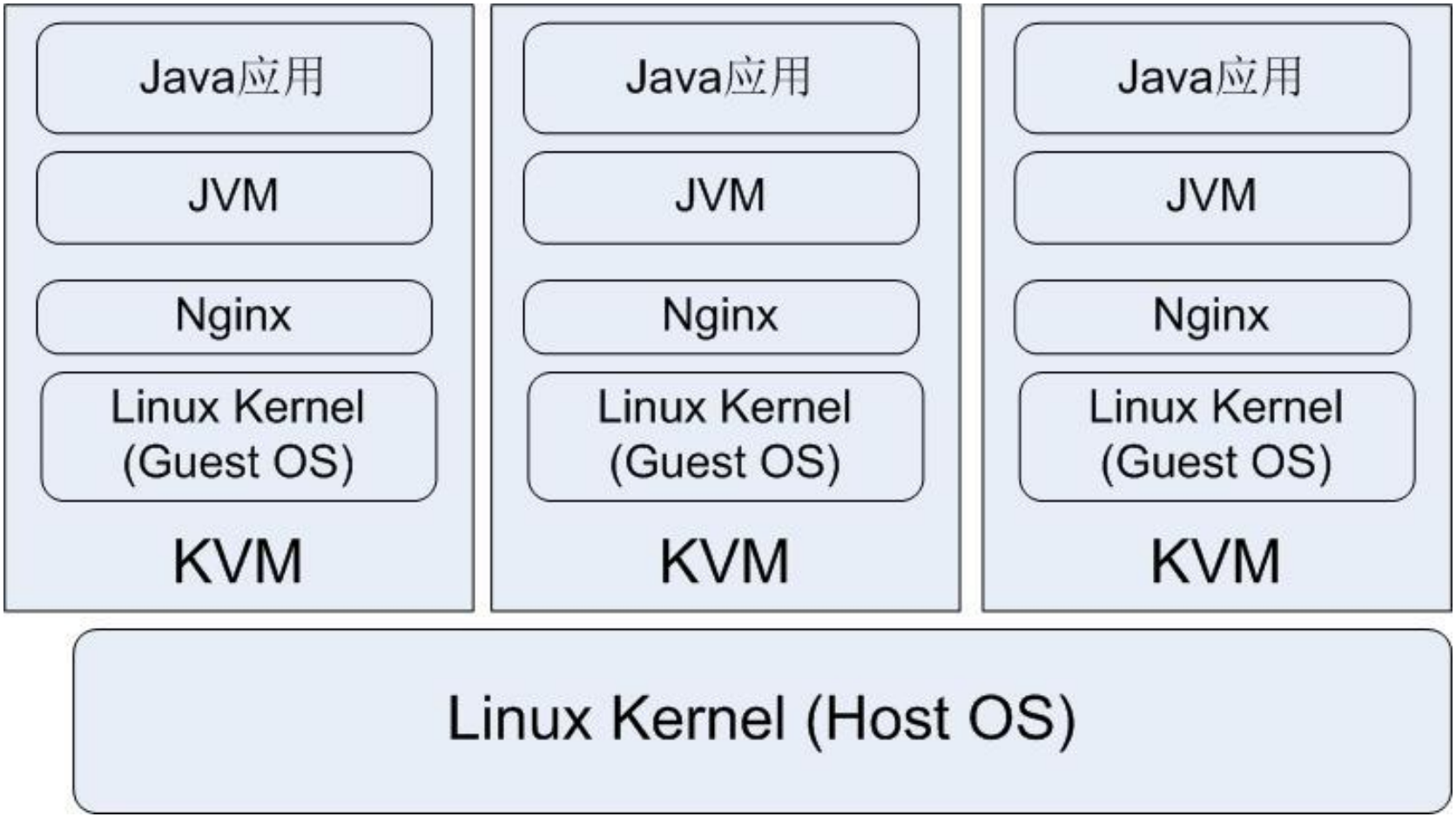
提纲

- 项目背景：主站的现状
- 选型的过程
- Cgroup/LinuxContainer介绍
- 定制和开发
- 存在的问题和对策

项目背景

- 主站：跑在xen虚拟机上的Java应用
  - 处理业务逻辑，本地无重要存储，无状态。
  - 一台物理机部署3台虚拟机
  - 双路Xeon，48GB内存
  - 多数应用是非交易相关的
  - 有1/3虚拟机峰值Load小于0.5
  - 基本无磁盘IO，主要资源需求集中于cpu和内存
  - 机器数量非常多

项目背景(续)


Java应用
JVM
Nginx
Linux Kernel
(Guest OS)
KVM
Java应用
JVM
Nginx
Linux Kernel
(Guest OS)
KVM
Java应用
JVM
Nginx
Linux Kernel
(Guest OS)
KVM
Linux Kernel (Host OS)

我们想要什么？

- 一台物理机跑更多虚拟机
- 更好的组合应用，做到集群利用率均衡
  - 消耗资源多的和少的部署在一起
  - 核心应用和非核心部署在一起
- 如果某台虚拟机流量上升需要更多资源，给它分配的内存和cpu也增加，反之亦然。
- 不增加监控、运维成本

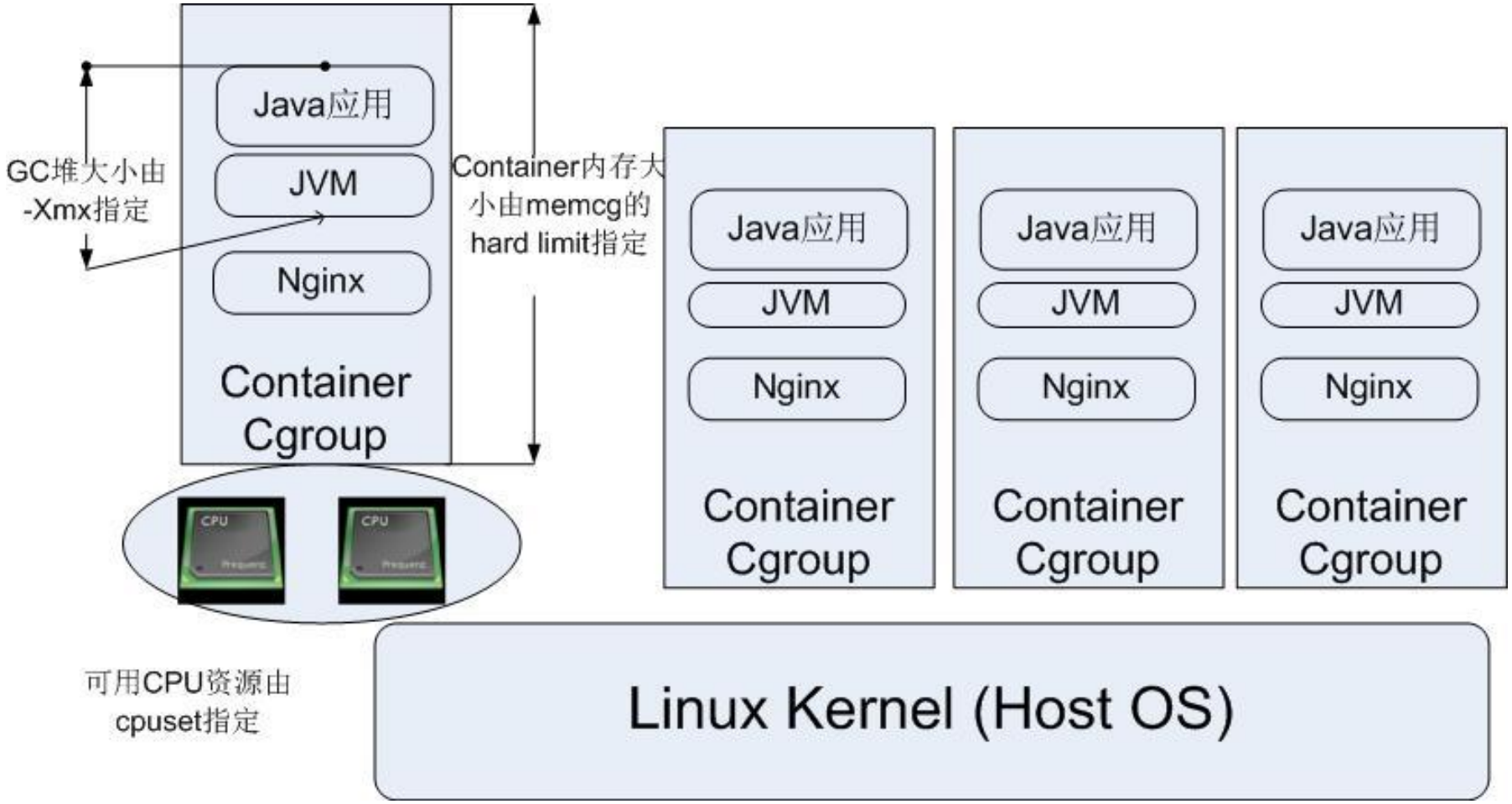
OS层次上的虚拟化

- Operating system-level virtualization
- 各种称呼, “containers”, “jails”, 增强的chroot
- Container是安全容器与资源容器的组合
  - 安全容器： chroot, UID/PID/IPC/Network Namespace
  - 资源容器： Control Cgroup (Cgroup)
- 实现
  - Linux
    - OpenVZ, Linux-VServer, LinuXContainer(LXC)
  - Solaris Zones
  - FreeBSD Jails

Cgroup

- LXC = Namespace + Cgroup
- 基于进程组的资源管理
- 基础框架 + 子控制器
  - 使用Cgroup的分组机制，对一组进程就某种系统资源实现资源管理。
- cpuset子控制器
  - 为进程组分配一组允许使用的CPU和内存结点
- memcg子控制器
  - 限制进程组允许使用的物理内存
  - 只限制匿名页和Page Cache，不包括内核自己使用的内存
  - 内存紧张时使用与全局回收同样的算法在组内回收
- Blkio子控制器
  - 限制进程组的磁盘IO带宽
  - 实际控制的是磁盘为各进程组服务的时间片数量，和进程调度器cfs针对cpu的分时服务原理相同

实际使用的方案


Java应用
JVM
Nginx
Container
Cgroup
GC堆大小由
-Xmx指定
Container内存大
小由memcg的
hard limit指定
CPU
CPU
可用CPU资源由
cpuset指定
Java应用
JVM
Nginx
Container
Cgroup
Java应用
JVM
Nginx
Container
Cgroup
Java应用
JVM
Nginx
Container
Cgroup
Linux Kernel (Host OS)

虚拟化方案比较

	动态迁移	CPU增减	内存增减
XEN(半虚拟化)	支持	通过Guest OS CPU热插拔机制支持	通过balloon driver支持
KVM(全虚拟化)	支持	通过Guest OS CPU热插拔机制支持	通过balloon driver支持
Linu***XC***ontainer (LXC)	不支持	通过cpuset和CFS组调度扩展支持	通过Cgroup支持

	速度	异构模拟	可运行不同OS	安全隔离性
XEN(半虚拟化)	比本地环境慢	支持	支持	强
KVM(全虚拟化)	比本地环境慢	支持	支持	强
LXC	同本地环境	不支持	不支持	差

- http://www.linux-kvm.org/page/CPUHotPlug
- http://virt.kernelnewbies.org/TechComparison
- http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines

Containers的优劣

- 优点
  - 虚拟化开销小，一台物理机跑很多“小”虚拟机
  - 通过Cgroup**增减**CPU/内存非常方便，调整速度很快
  - 和本地环境相同的速度
- 缺点
  - 不能热迁移➔ 调度流量，而不是调度机器
  - 不能模拟不同体系结构、装不同os➔ 不需要
  - 安全隔离差➔ 内部集群，由运维操作
  - ForkBomb、优先级反转、Cgroup实现缺陷➔ 见招拆招
    - Memcg不能控制buffer和内核内存
  - Proc下的多数文件不认识Container，top/sar/free/iostat都用不了➔ 自己按需求修改内核

性能监控

- /proc没有“虚拟化”
- 为每个Container计算Load和CPU利用率
  - 社区已有部分Patch，但没被接受
  - Upstream内核中为了提高SMP伸缩性，计算Load变得非常复杂—而且有Bug，幸好我们目前不需要
  - 如何表示与自己共享CPU的其他Container占用的CPU时间？借用半虚拟化中的steal time概念
- 为每个Container维护/proc/meminfo
  - 社区已有部分Patch，但仍然不被接受
  - Buffer列只能显示零
  - 内存总量即为Cgroup允许的该容器内存上限

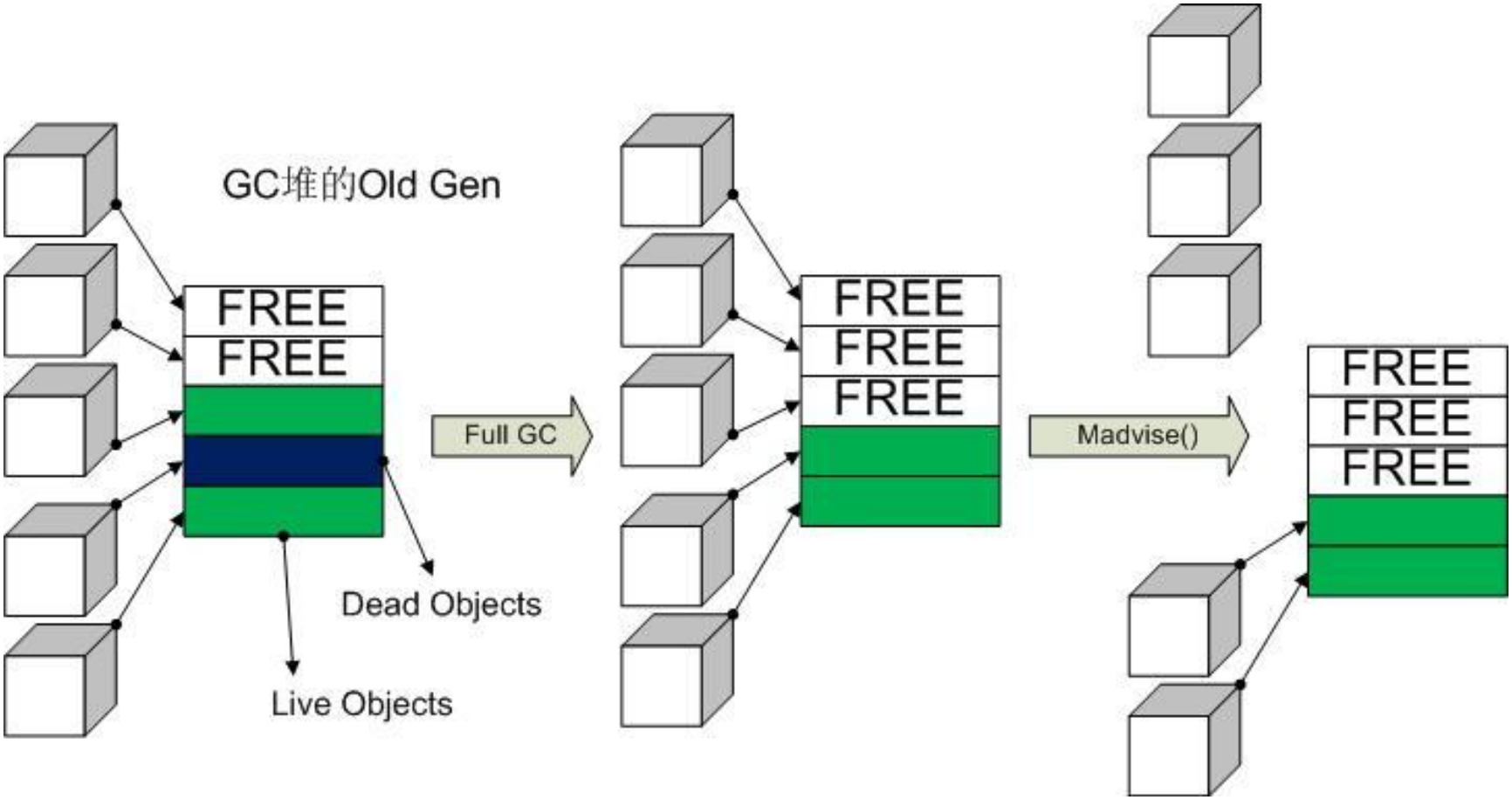
弹性内存分配

- 重新思考最基础的设计
- Hotspot 的GC堆大小(-Xmx)和mem cgroup的上限(hard limit)如何取值？堆内和堆外内存应该给多少？
- 内存满了会发生什么？
  - 匿名页被交换到磁盘上
  - Page cache页直接丢弃，脏页丢弃前先回写
  - 部署大量Java应用时内存主要是GC堆的匿名页
  - 垃圾所占匿名页从其失去最后一个引用之时起到下次GC之时止不会被访问，被交换出去的概率相对更大
  - 把垃圾交换到磁盘上是一种浪费

弹性内存分配

- Cgroup可以动态调整内存上限，但Hotspot的GC堆大小是启动时设定的
- 基本想法：在内存紧张时，“摘掉”GC堆中垃圾对象所占的物理内存
  - 内存紧张可能由于全局内存缺乏，也可能由于cgroup组内内存不足
  - 仍然不能让GC堆扩大，但可以一开始就设个比较大的值(overcommit)，在资源紧张时让其缩小
  - 对GC堆上限(-Xmx)和memcg上限(hard limit)这两个参数不再敏感

弹性内存分配(续1)


GC堆的Old Gen
FREE
FREE
Dead Objects
Live Objects
Full GC
FREE
FREE
FREE
Madvise()
FREE
FREE
FREE

很多细节

- 有许多Container，该回收谁的内存?
  - Shares: Container的权重，我们在这里使用其允许使用的最大内存总量做为权重
  - Pages: Container实际使用的内存量
  - 回收倾向 = Shares / Page，越小越容易被选中
- 希望的行为
  - Container活跃时得到允许范围内尽可能多内存
  - Container的工作集缩小(不活跃)时剥夺它的内存
  - 引入内存活跃程度百分比α，通过定期扫描页得到
  - 引入不活跃内存惩罚系数idle_tax $\in [1, +\infty)$
  - 回收倾向 = Shares / Pages * (α + idle_tax * (1 – α))
- 算法来自Vmware ESX Server

遗留问题

- 内存还剩多少时触发?
  - Full GC时Young Gen中活对象先整体提升，所以内存使用量可能出现先升后降的现象
  - Full GC用时远长于一次kswapd活动，内存用量在Full GC期间可能继续上涨至direct reclaim
  - 必须考虑和Page cache用量的平衡
- 目前只支持ParallelGC collector，不支持CMS
- numa aware
  - 目前实现把每个numa结点视为单独的机器计算
  - 然而ParallelGC collector不是numa-aware的，被某个内存紧张的结点挑中的Container回收到的内存未必在这个结点上
- 内存压力有可能来自于应用内存持续泄露，此时oom比触发GC来回收内存更好

结束

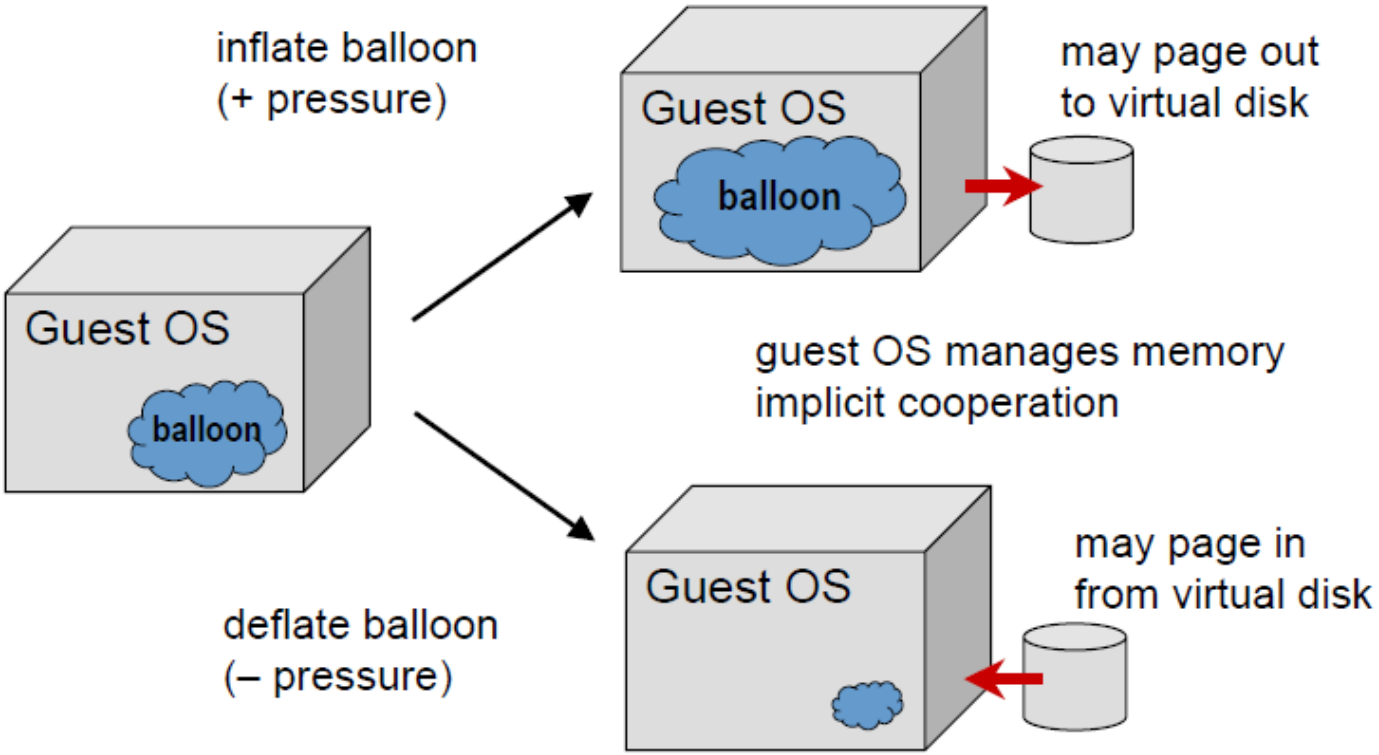
- 谢谢!
- 提问时间

Backup/1

- Upstream内核中Load计算不准的Bug
  - [Patch] sched: Cure load average vs NO_HZ woes
  - [Patch] sched: Cure more NO_HZ load average woes
  - [Patch] sched: Fix nohz load accounting – again!
  - [PATCH] sched: Folding nohz load accounting more accurate

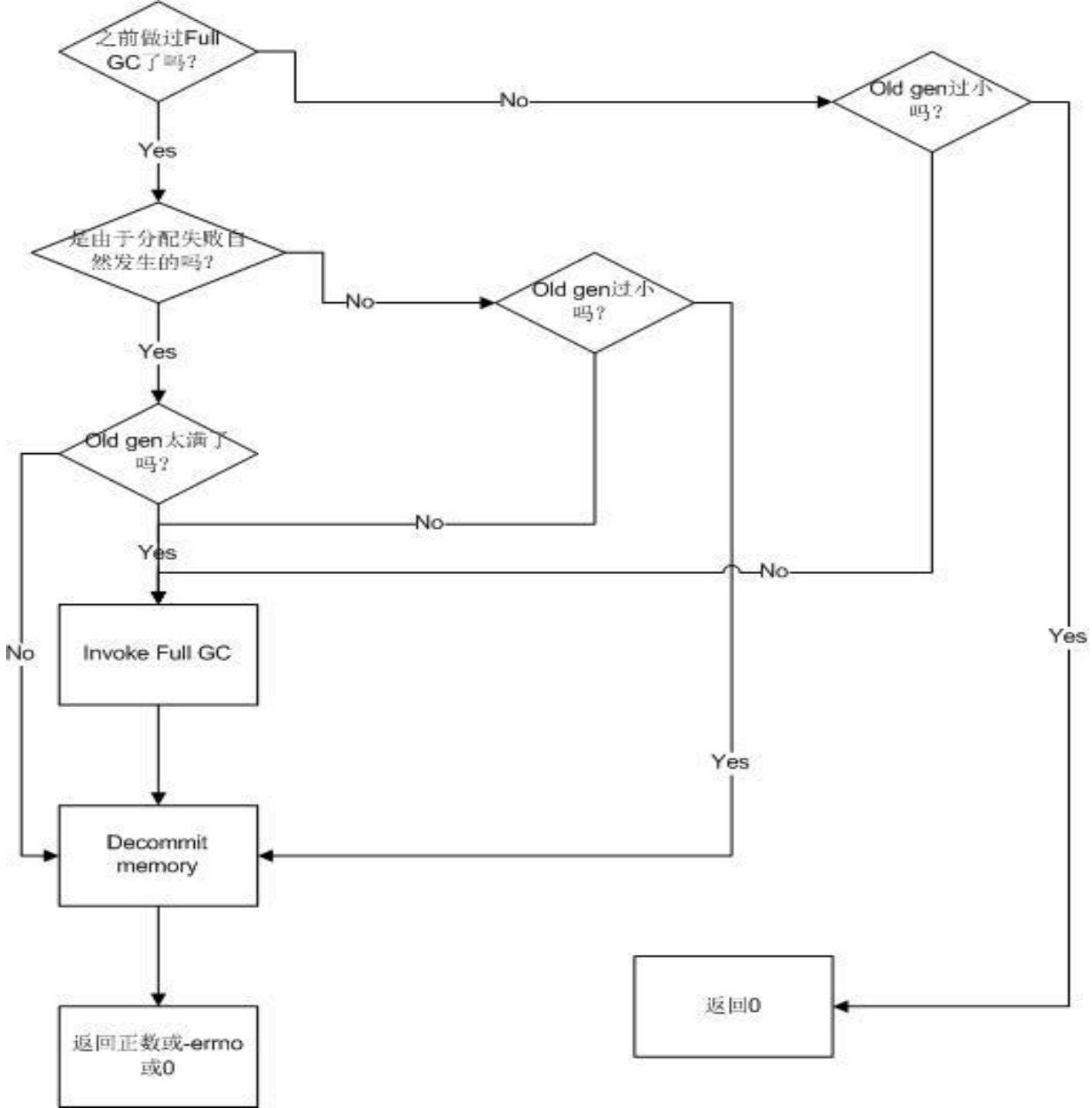
Backup/2

Ballooning



弹性内存分配实现

- 一个内核线程定期扫描各Container的内存使用情况，得到内存活跃程度百分比α
  - [patchset] idle page tracking / working set estimation from Google，在此基础上修改
- 使用Eventfd通知用户态程序
- 一个用户态伺服程序，用来接通知并转发给Hotspot JVM
- kernel patch changed lines约3k行
- 修改Hotspot，通过Attach API实现触发Full GC和使用madvise()释放内存(需要考虑很多细节)
- 约700行Hotspot patch


之前做过Full GC了吗?
No
Old gen过小吗?
Yes
是由于分配失败自然发生的吗?
No
Old gen过小吗?
Yes
Old gen太满了吗?
No
Yes
No
Yes
Invoke Full GC
Yes
Decommit memory
返回0
返回正数或-errno或0

参考

- “Memory Resource Management in Vmware ESX Server”, Carl Waldspurger