

JDJDK

可控与高效的JDK

臧琳

京东零售-数据中台-数据基础平台部-平台架构部-高性能集群研发部

zanglin5@jd.com

2019-03-20

www.jd.com



Contents



01

JDK8 vs JDJDK

02

回顾 G1 vs CMS

03

痛点list & JDJDK解决方案

04

JDJDK 展望与未来



JDK8 vs JDJDK

JDK 8 vs JDJDK

- 30+ JDK重要特性

- ▶ Runtime Optimizations

- ▶ Improve Contended Locking
 - ▶ App CDS
 - ▶ ...

- ▶ Compiler optimizations

- ▶ Segmented Code Cache
 - ▶ JVMCI
 - ▶ AOT
 - ▶ ...

- ▶ GC Optimizations

- ▶ Unified GC logging
 - ▶ Make G1 default
 - ▶ G1 parallel Full GC
 - ▶ ...

- ▶ Tooling/Debugging

- ▶ More tools
 - ▶ Stack walking API
 - ▶ Unified VM logging
 - ▶ JFR

- ▶ 自研特性

- ▶ 3 JDK12重要特性

- ▶ 2 G1 optimizations
 - ▶ 1 Default CDS

- ▶ 性能优化

- ▶ 更适合大堆的G1

- ▶ 工具增强

- ▶ Jmap

- ▶ 功能拓展

- ▶ 定时GC
 - ▶ 内存管理

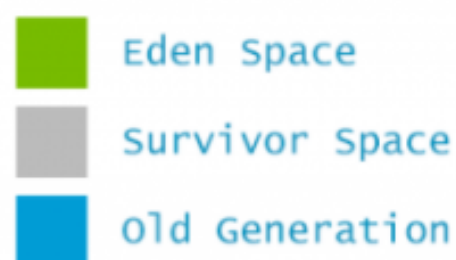
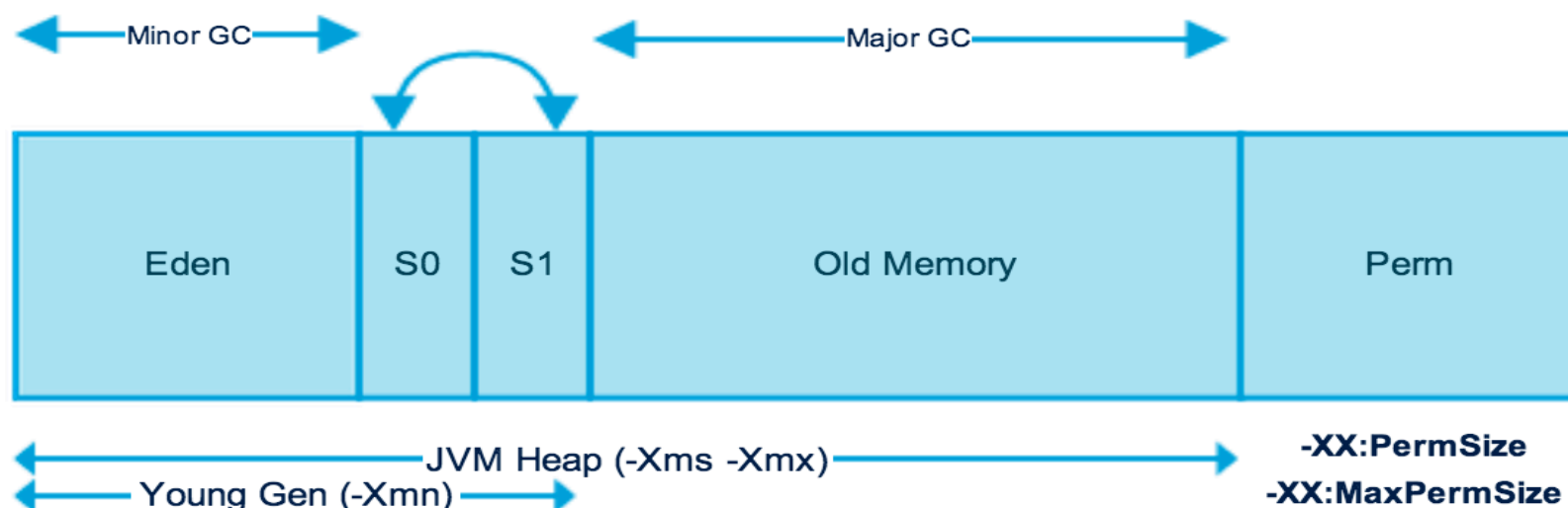


回顾 G1 vs CMS

回顾：G1 VS CMS

► CMS

- 年轻代
 - 连续
 - 固定
- 老年代
 - 并行非拷贝
 - 碎片化



► G1

- 年轻代
 - 连续不连续
 - 不固定
- 老年代
 - 连续不连续
 - 碎片低

回顾：G1 VS CMS

	CMS	G1	备注
分配算法	Bump Pointer + TLAB	Bump Pointer + TLAB	
YoungGC	Parallel Copying + promotion	Regional Parallel Copying + promotion	G1优势： 基于region的批处理
MajorGC (old GC/ MixGC)	Whole old Gen Concurrent marking No copy	Partial old gen Concurrent marking copying	G1优势： 回收目标可配置， 暂停时间可尽量保证， 最小化碎片问题
Full GC	STW Mark Compact	STW parallel Mark Compact	G1优势： 并行压缩算法， 效率高，暂停短
综合	高效分配， 回收目标不可配置， 暂停时间不可控， 老年代碎片化， Heap利用率低 FullGC易发生	高效分配， 可动态配置的回收目标 暂停时间可尽量保障 无碎片化问题 Heap利用率高 FullGC几乎不发生	G1已经取代CMS成为JDK 的默认垃圾收集器 开源社区对于CMS不会在 做进一步升级和调优



痛点list & JDJDK解决方案

痛点 list

- Full GC 和 Old GC
- 令人头疼的OOM
- JVM 内存占用
- GC停顿时间
- JVM中的玄学 – 参数调优
- 大堆下的分配性能提升
- Jmap 堆信息工具

Full GC And Old GC

- Full GC
 - 相见不如怀念, 怀念不如不见
 - OOM 前的最后一根稻草
 - G1/CMS Full GC
 - » 回收量最大, 最全
 - » 回收所有垃圾
 - 停顿时间最长
 - 轻则造成系统抖动, 重则系统崩溃
 - HDFS NN 上 180GB Heap 一次Full GC直接造成进程长时间假死
 - 禁止application做FullGC -XX:+DisableExplicitGC
 - Old GC
 - OldGC 并发 + 部分收集 / 停顿 + 部分收集
 - 过犹不及, 难以控制
 - 过多 -> 系统抖动, 影响YGC
 - 过少 -> Full GC

Full GC And Old GC



- OpenJDK
 - 参数调优 (见后)
 - 缺少有效的机制
- JDJDK - 在对的时间让对的事情发生
 - G1 - 定时 Old GC <可选GC工作量/可选FullGC>
 - 减少不可控的GC发生的几率和力度
 - 动态参数
 - jcmd vm.set_flag G1PeriodicGCForceSpanBeginAt 18:00
 - jcmd vm.set_flag G1PeriodicGCForceInterval 20000
 - jcmd vm.set_flag G1PeriodicGCForceSpanLengthInMin 3
 - jcmd vm.set_flag G1PeriodicGCForce true

令人头疼的OOM & FullGC



- 行为
 - Allocation -> YGC ...->Old GC...->YGC..->...Full GC->OOM
- OOM
 - Heap需求突然增加
- Full GC
 - 尽管有定时GC，仍存在几率
 - Object 突然井喷
- OpenJDK
 - 加大Xmx重启
- JDJDK
 - 90% Heap利用率预警
 - 动态增长内存：
 - -XX:ExtensibleHeapRatio = 20
 - Jcmd <pid> VM.set_flag UseExtensibleHeap true
 - 不用不发力

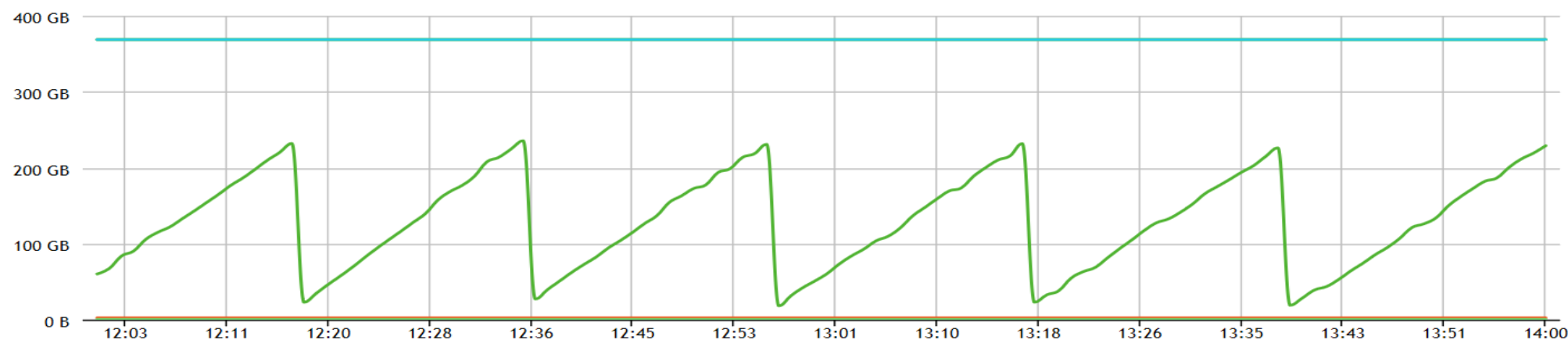
JVM 内存占用



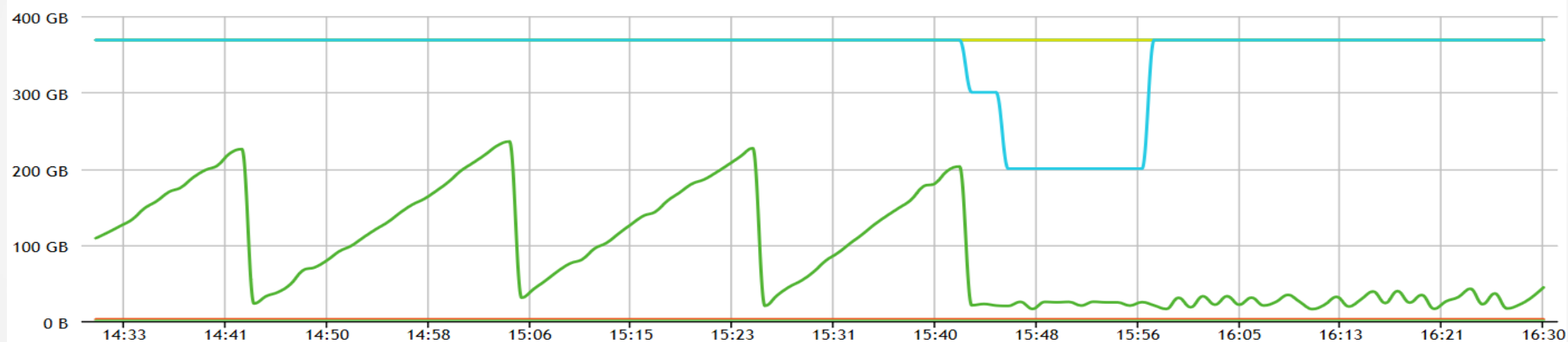
- Xmx Xms
 - 性能还是存储
- Used, committed, Capacity
- 是否承诺了太多，还是原本给的就不够？
 - Xmx 大 -> 避免OOM / 系统资源浪费， Native需求不够
 - Xmx 小 -> FullGC/OOM
 - 实际heap使用率可能保持在一定的范围之内
 - 特定情况下，我们需要JVM“让”出一部分内存
 - OpenJDK
 - 又要性能又要存储，我也没有办法啊
 - JDJDK
 - XX:JDSwitchesHeapControlCeil=2000 (M)



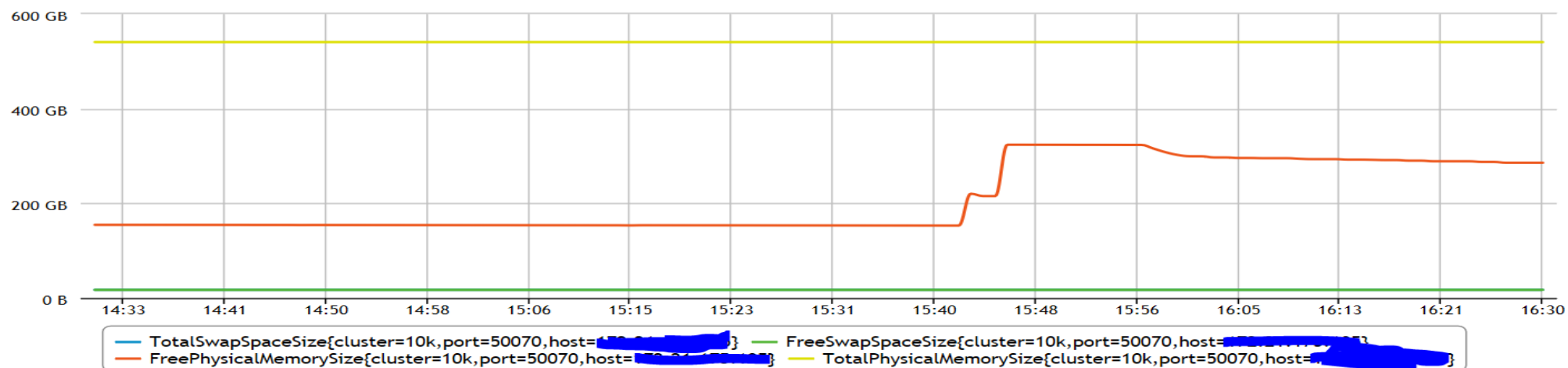
NameNode[] 堆内存使用量



NameNode[] 堆内存使用量

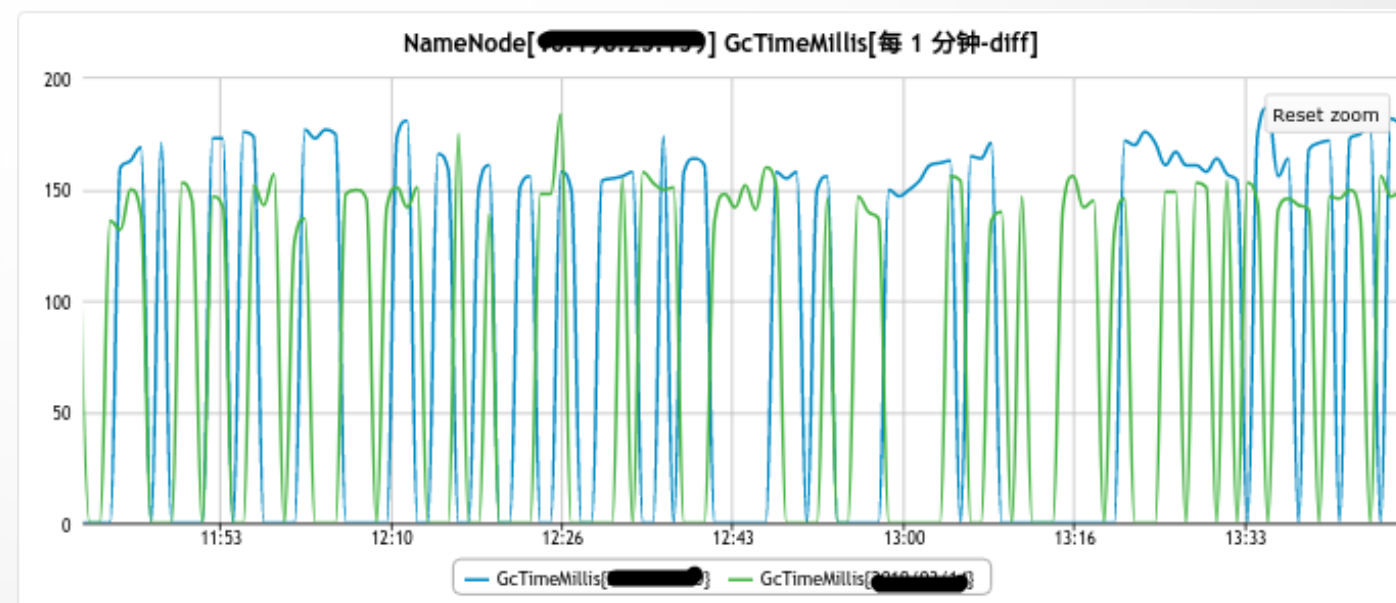
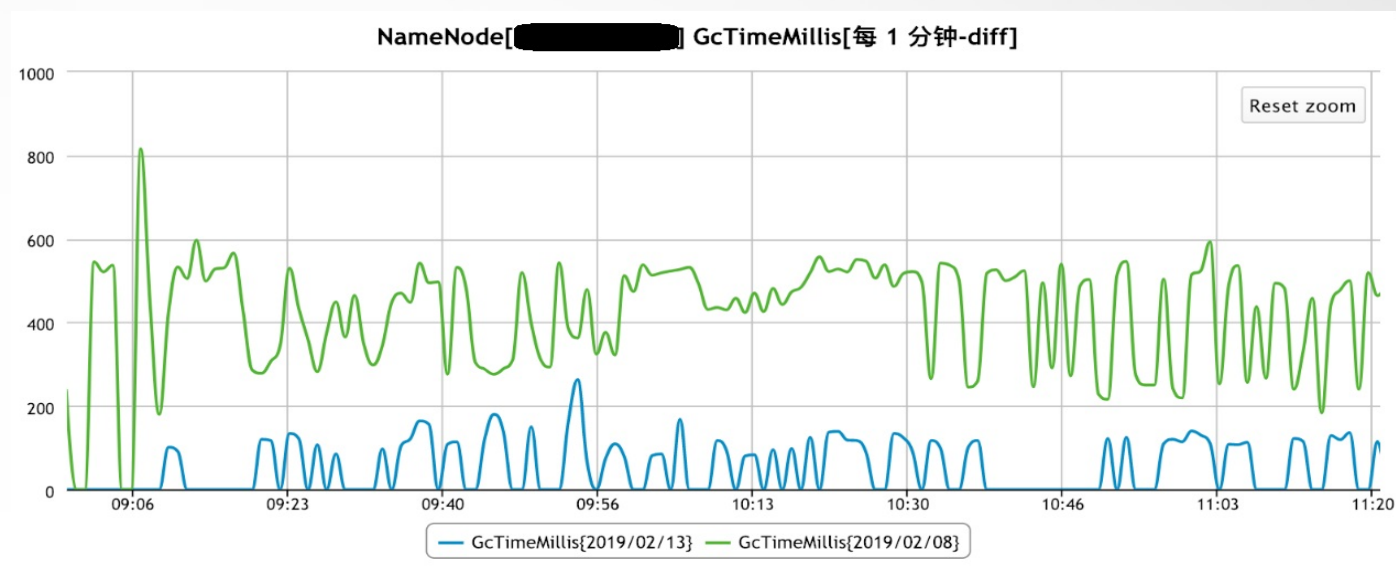


NameNode[] 系统资源使用

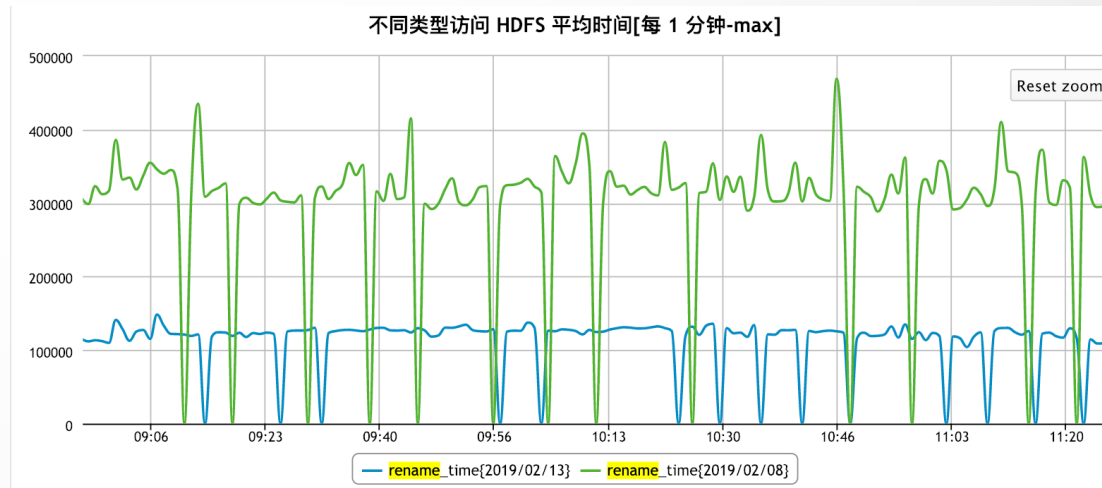
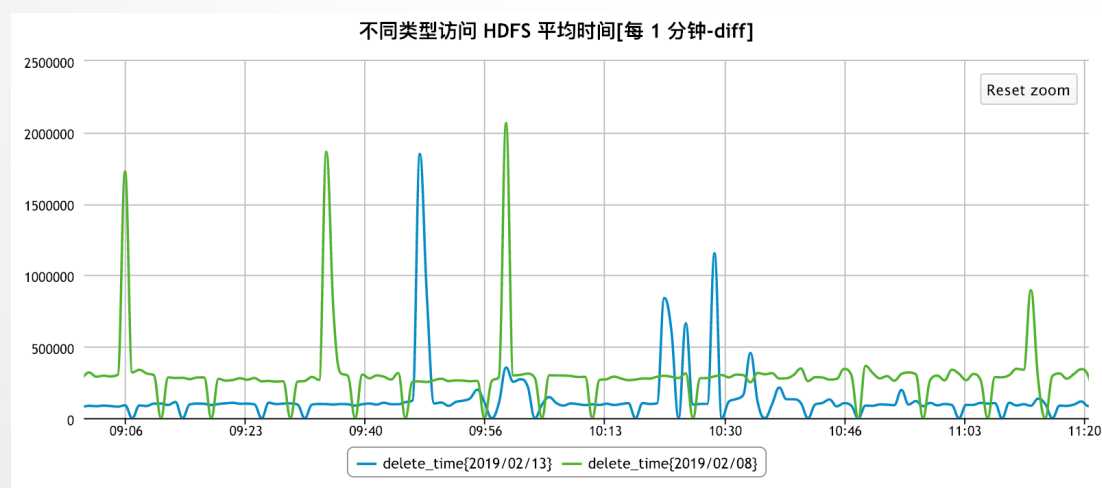
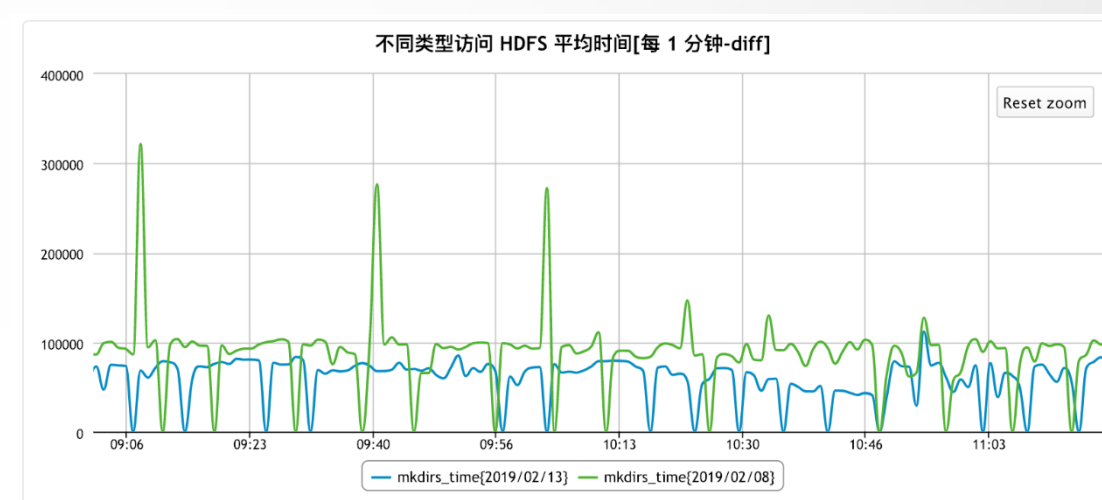
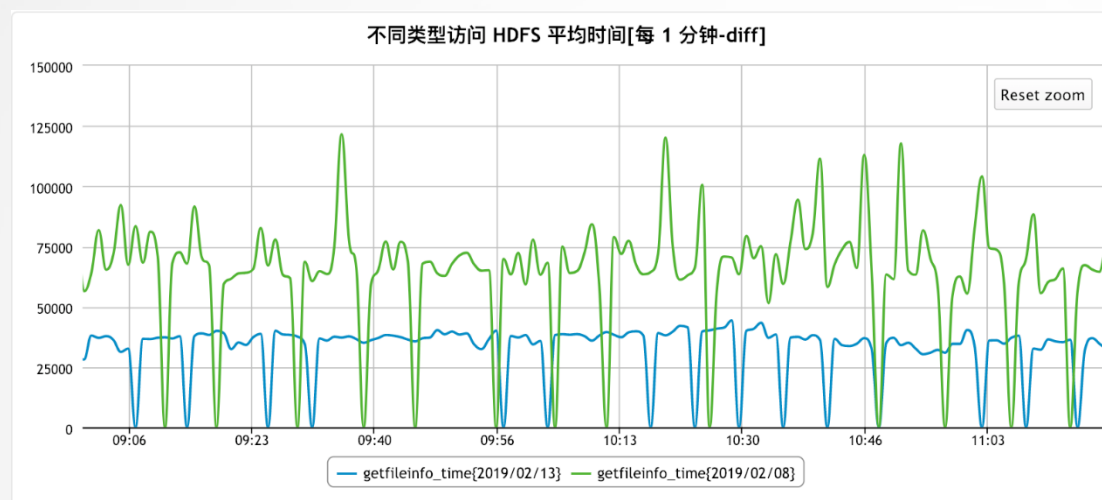


GC 停顿时间

- 别让我等太久
- 停顿时间
 - G1 VS CMS
 - Openjdk VS JDJDK
- JDJDK
 - 在JDK12新feature
 - 结合定时GC
 - 自研优化



大堆下分配性能



一顿操作猛如虎 - 参数调优



- 怎么调参减少GC?
- YGC频繁怎么办
- 该不该触发Old GC?
- 啥是tenringThreshold
- ParallGCThread = ?
- concGCThread = ?
- Xmx Xms Xmn
- AlwaysPretouch
- CMSxxxx

Young space tenuring	
-XX:InitialTenuringThreshold=8	Initial value for tenuring threshold (number of collections before object will be promoted to old space)
-XX:MaxTenuringThreshold=15	Max value for tenuring threshold
-XX:PretenureSizeThreshold=2m	Max object size allowed to be allocated in young space (large objects will be allocated directly in old space). Thread local allocation bypasses this check, so if TLAB is large enough object exceeding size threshold still may be allocated in young space.
-XX:+AlwaysTenure	Promote all objects surviving young collection immediately to tenured space (equivalent of -XX:MaxTenuringThreshold=0)
-XX:+NeverTenure	Objects from young space will never get promoted to tenured space unless survivor space is not enough to keep them
Thread local allocation	
-XX:+UseTLAB	Use thread local allocation blocks in eden
-XX:+ResizeTLAB	Let JVM resize TLABs per thread
-XX:TLABSize=1m	Initial size of thread's TLAB
-XX:MinTLABSize=64k	Min size of TLAB
Parallel processing	
-XX:ConcGCThreads=2	Number of parallel threads used for concurrent phase.
-XX:ParallelGCThreads=16	Number of parallel threads used for stop-the-world phases.
-XX:+ParallelRefProcEnabled	Enable parallel processing of references during GC pause
Miscellaneous	
-XX:+DisableExplicitGC	JVM will ignore application calls to System.gc()
-XX:+ExplicitGCInvokesConcurrent	Let System.gc() trigger concurrent collection instead of full GC
-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses	Same as above but also triggers permanent space collection.
-XX:SoftRefLRUPolicyMSPerMB=1000	Factor for calculating soft reference TTL based on free heap size
-XX:OnOutOfMemoryError=...	Command to be executed in case of out of memory. E.g. "kill -9 %p" on Unix or "taskkill /F /PID %p" on Windows.

Concurrent Mark Sweep (CMS)		Garbage First (G1)	
CMS initiating options		G1 initiating options	
-XX:+UseCMSInitiatingOccupancyOnly	Only use predefined occupancy as only criterion for starting a CMS collection (disable adaptive behaviour)	-XX:G1HeapRegionSize=32m	Size of heap region
-XX:CMSInitiatingOccupancyFraction=70	Percentage CMS generation occupancy to start a CMS cycle. A negative value means that CMSTriggerRatio is used.	-XX:G1ReservePercent=10	Percentage of heap to keep free. Reserved memory is used as last resort to avoid promotion failure.
-XX:CMSBootstrapOccupancy=50	Percentage CMS generation occupancy at which to initiate CMS collection for bootstrapping collection stats.	-XX:InitiatingHeapOccupancyPercent=45	Percentage of (entire) heap occupancy to trigger concurrent GC
-XX:CMSTriggerRatio=70	Percentage of MinHeapFreeRatio in CMS generation that is allocated before a CMS collection cycle commences.	-XX:G1MixedGCCountTarget=8	Target number of mixed collections after a marking cycle
-XX:CMSTriggerInterval=60000	Periodically triggers CMS collection. Useful for deterministic object finalization.	-XX:G1HeapWastePercent=10	If garbage level is below threshold, G1 will not attempt to reclaim memory further
CMS Stop-the-World pauses tuning		-XX:G1ConfidencePercent=50	Confidence level for MMU/pause prediction
-XX:CMSWaitDuration=30000	Once CMS collection is triggered, it will wait for next young collection to perform initial mark right after. This parameter specifies how long CMS can wait for young collection	-XX:MaxGCPauseMillis=500	Target GC pause duration. G1 is not deterministic, so no guarantees for GC pause to satisfy this limit.
-XX:+CMSScavengeBeforeRemark	Force young collection before remark phase	CMS Diagnostic options	
-XX:+CMSScheduleRemarkEdenSizeThreshold	If Eden used is below this value, don't try to schedule remark	-XX:PrintCMSStatistics=1	Print additional CMS statistics. Very verbose if n=2.
-XX:CMSScheduleRemarkEdenPenetration=20	Eden occupancy % at which to try and schedule remark pause	-XX:+PrintCMSInitiationStatistics	Print CMS initiation details
-XX:CMSScheduleRemarkSamplingRatio=4	Start sampling Eden top at least before young generation occupancy reaches 1/ of the size at which we plan to schedule remark	-XX:+CMSDumpAtPromotionFailure	Dump useful information about the state of the CMS old generation upon a promotion failure
CMS Concurrency options		-XX:+CMSPrintChunksInDump	(with optin above) Add more detailed information about the free chunks
-XX:+CMSParallelInitialMarkEnabled	Whether parallel initial mark is enabled (enabled by default)	-XX:+CMSPrintObjectsInDump	(with optin above) Add more detailed information about the allocated objects
-XX:+CMSParallelRemarkEnabled	Whether parallel remark is enabled (enabled by default)	Misc CMS options	
-XX:+CMSParallelSurvivorRemarkEnabled	Whether parallel remark of survivor space enabled, effective only with option above (enabled by default)	-XX:+CMSClassUnloadingEnabled	If not enabled, CMS will not clean permanent space. You may need to enable it for containers such as JEE or OSGi.
-XX:+CMSConcurrentMTEnabled	Use multiple threads for concurrent phases.	-XX:+CMSIncrementalMode	Enable incremental CMS mode. Incremental mode was meant for servers with small number of CPU, but may be used on multicore servers to benefit from more conservative initiation strategy.
		-XX:+CMSOldPLABMin=16 -XX:+CMSOldPLABMax=1024	Min and max size of CMS gen PLAB caches per worker per block size
		Options for "deterministic" CMS, they disable some heuristics and require careful validation	

验证一看原地杵

举个栗子（案例）

- 问题

“ParNew+cms上线后发现不再有每小时定时full gc的情况，但是运行几天后会出现应用的tp99逐步升高的情况”

- ParNew+CMS
- No FullGC
- Tp99 slow down

- 猜测:

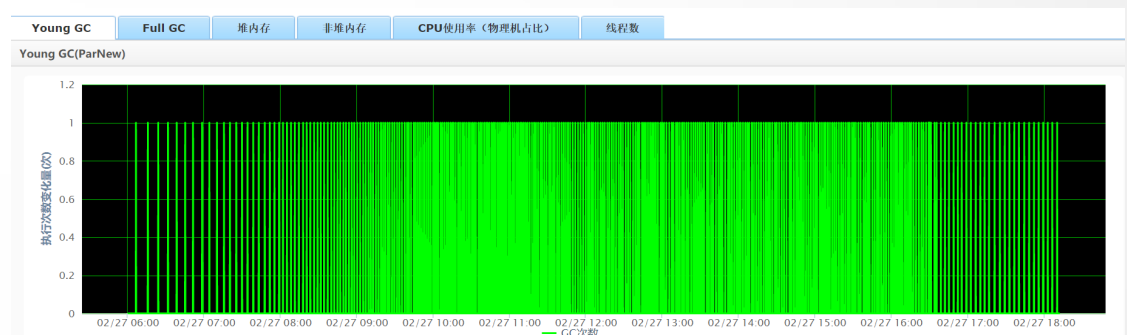
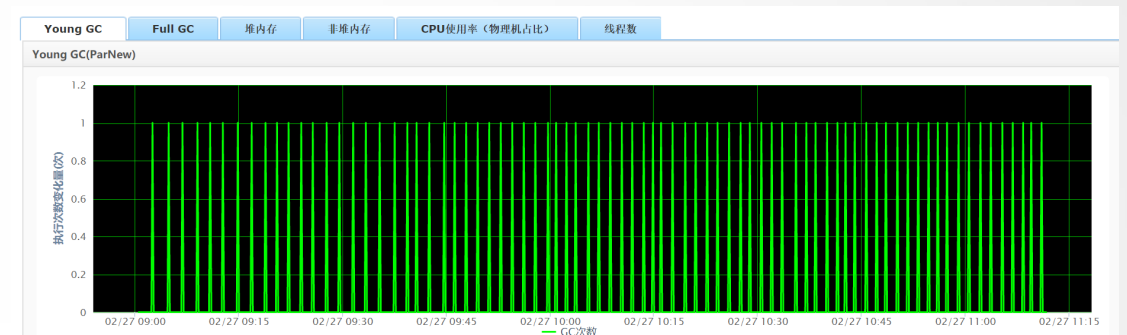
- YGC 越来越频繁
- 或者 OLD GC越来越多

- 怎么确定呢?

- gclog + 监控

案例 (续)

- Gclog显示
 - 一直没有出现 Old GC
 - YGC 频繁
 - 高峰时段100ms发生一次
 - Old space 增长缓慢
 - 每15-18小时 增长100MB
 - 推断任务特点
 - 短/多/快
 - “响应时间都在5ms以内的实时服务”
- 监控显示YGC频率逐渐增高
 - why



案例（续）

- 启动参数

- -Xms4096m -Xmx4096m -XX:MaxPermSize=512m -Xmn1024M -XX:PermSize=128M
- -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+UseCMSInitiatingOccupancyOnly -XX:+CMSClassUnloadingEnabled
- -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:/export/Logs/sopocs.ofc.jd.com/gc.log
- -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/export/Instances/sopocs/server1/logs XX>ErrorFile=/export/Instances/sopocs/server1/logs/java_error_%p.log

- 结论

- 没发生oldGC 所以和old 没关系 吗?
- YGC频率高, 考虑YGC

- 建议

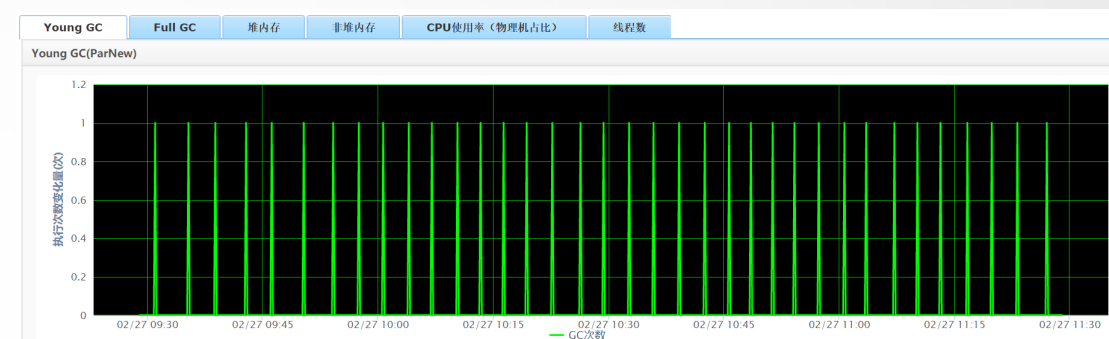
- 增大Xmn 加大YGC间隔

一顿操作猛如虎!

案例（续）

- 结果

- Xmn=2000m
- YGC频率确实低了
- Tp99趋势没变
- 和xmn=1000m看起来，表现差不多



- 详细现象

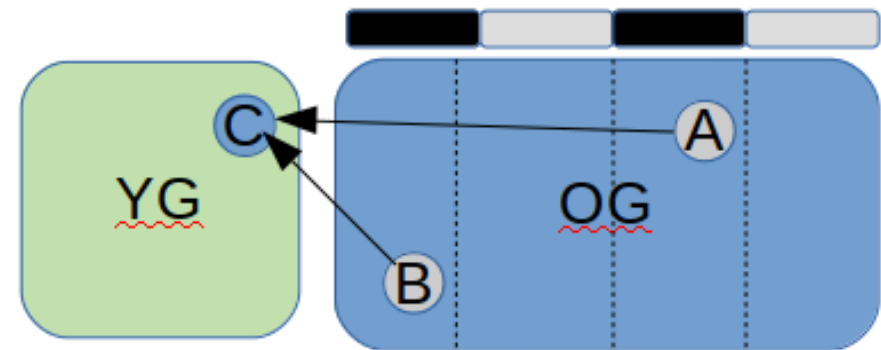
- 系统刚启动的时候，tp99在4ms左右，运行多天后，达到20ms
- YGC 越来越频繁
- 始终没有old GC
 - -XX:CMSInitiatingOccupancyFraction=70
 - -XX:+UseCMSInitiatingOccupancyOnly
 - old space增长缓慢
 - 难道是old gc的原因？

测试结果原地杵

案例 (续)

- 分析

- Old object 不被回收
 - 跨代引用积累
 - Card table 扫描耗时
 - YGC越来越长 越来越频繁
- 触发老年代gc
 - 性能影响 待查
- 验证
 - GC 日志分析

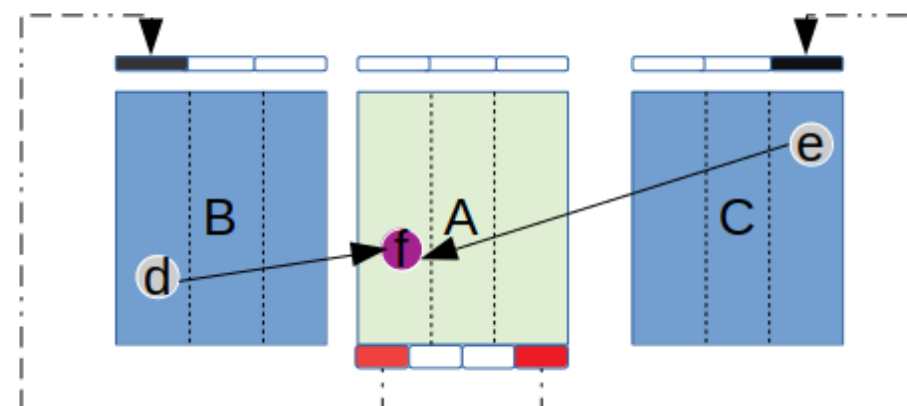
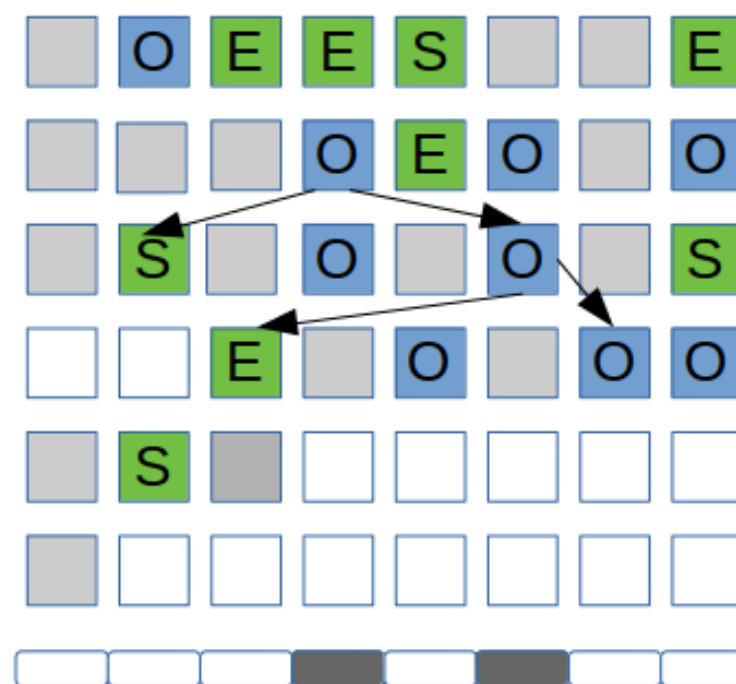
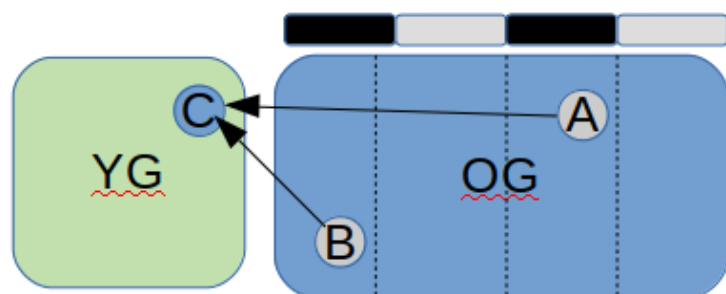


- 结果

- -XX:CMSInitiatingOccupancyFraction=10 (old gen达到100M左右触发OLDGC)
- -XX:MaxTenuringThreshold=15 (尽量晚 (少) 把对象加入老年代)
- 观察结果大概1天半触发一次old gc
- Tp99稳定

案例 (续)

G1 怎么样?



其他重要特性

- Heap
 - 180g → 360g
 - fullGC
 - 文件数承载能力从4亿上升到10亿
 - 系统更加稳定

JDJDK 展望

- 你将看到.....
- 高度优化的JVM
 - 半自治堆
 - 智能自适应调优系统
 - 丰富的线上处理工具集
 - Profiling
 - Debugging
 - Gceasy/fastthread... internal version
 - Heap dump analyzer for extrem large heap
 - Java flame graph
 -

关于JDJDK

JDJDK 试用版：

<http://blog.jd.local/bbs/article/1552622568648>

JDJDK issues & 需求：（项目名： JDJDK）

<http://jira.jd.com/secure/Dashboard.jspa>

JDJDK项目源码：

<http://source.jd.com/app/JDJDK>

联系我们：

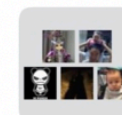
京东JDK交流群： 81523801 （timeline）

微信二维码：



咚咚: 81523801

THANKS



京东JDK交流群



该二维码7天内(3月26日前)有效, 重新进入将更新

请美美的使用JDK, 烦心事都交给我们